NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

OPTIMAL SIZE OF JOB POOL FOR INITIATING A SCHEDULING EVENT

by

James M. Breitinger

September 1999

Thesis Advisor: Taylor Kidd
Second Reader: Debra Hensgen

Approved for public release; distribution is unlimited.

19991129 005

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1999	Master's Thesis

4. TITLE AND SUBTITLE
OPTIMAL SIZE OF JOB POOL FOR INITIATING A SCHEDULING EVENT

5. FUNDING NUMBERS

6. AUTHOR(S)
Breitinger, James M.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

In today's military with its dwindling resources, making the best use of computers, particularly to support real-time commercial off-the-shelf (COTS) applications, is becoming critical for success. Resource Management Systems (RMS) strive to address this issue. The RMS's job scheduler is needed to ensure good quality of service (QoS) to all applications. This research uses discrete event simulation experiments to investigate the cost tradeoff between improving system performance through grouping incoming jobs to create better schedules, versus both (1) the time spent waiting for the group to accumulate and (2) the additional cost of computing schedules involving more jobs. A MaxMin $O(MN^2)$ greedy scheduling algorithm attempting to minimize the total time in system was used in these experiments. We analyzed the data generated from numerous experiments that used typical input parameters. As a result of this effort, we conclude that job grouping should be used when the utilization factor for the system is near 1.0, or precisely when the mean arrival rate is comparable to the total mean service rate of the processors. At this utilization rate, the group size should be equal to the number of machines in the system. However, when the utilization factor is significantly different from 1.0, each job should be scheduled as it arrives.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Heterogeneous Computing, Resource Management System, JAVA, Modeling and Simulation, Scheduling, Discrete Event Simulation, Quality of Service, RMS, QoS	139
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

OPTIMAL SIZE OF JOB POOL FOR INITIATING A SCHEDULING EVENT

James M. Breitinger
Major, United States Marine Corps
B.S., The Pennsylvania State University, 1987
MBA, National University, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1999

Author: _____
James M. Breitinger

Approved by: _____
Taylor Kidd, Thesis Advisor

_____
Debra Hensgen, Second Reader

_____
Dan Boger, Acting Chairman,
Computer Science Department

ABSTRACT

In today's military with its dwindling resources, making the best use of computers, particularly to support real-time commercial off-the-shelf (COTS) applications, is becoming critical for success. Resource Management Systems (RMS) strive to address this issue. The RMS's job scheduler is needed to ensure good quality of service (QoS) to all applications. This research uses discrete event simulation experiments to investigate the cost tradeoff between improving system performance through grouping incoming jobs to create better schedules, versus both (1) the time spent waiting for the group to accumulate and (2) the additional cost of computing schedules involving more jobs. A MaxMin $O(MN^2)$ greedy scheduling algorithm attempting to minimize the total time in system was used in these experiments. We analyzed the data generated from numerous experiments that used typical input parameters. As a result of this effort, we conclude that job grouping should be used when the utilization factor for the system is near 1.0, or precisely when the mean arrival rate is comparable to the total mean service rate of the processors. At this utilization rate, the group size should be equal to the number of machines in the system. However, when the utilization factor is significantly different from 1.0, each job should be scheduled as it arrives.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENT

The author would first like to thank his supportive family: Linda, Dustin and Haylee for their understanding and patience. Additionally, he thanks Dr. Taylor Kidd for his support during the work in performing this investigation and Mr. Shoukat Ali, Purdue University, for providing the information and algorithm concerning the generation of a HiLo, inconsistent ETC matrix necessary to complete the research for this thesis. Finally, thanks to the entire MSHN team. The sense of camaraderie and teamwork gave this research meaning and made the process endurable.

I. INTRODUCTION

This thesis investigates questions associated with obtaining the best Quality of Service (QoS) achievable in a distributed, heterogeneous computing (HC) environment. A necessary component within a HC environment is determining upon which resources to place and at what time to execute each job in order to optimize the QoS that all jobs obtain. In a typical environment, job requests arrive according to some random distribution. Immediately assigning jobs to resources, as they arrive, does not always result in the best performance in a heterogeneous environment (see [KIDD96]). Therefore, this thesis strives to determine the optimal point, in time, at which to schedule these jobs. Important considerations include whether this point is a function of the number of jobs that have been pooled, a function of the amount of time that has passed, or even a function of the resources that are currently available. The jobs can be interactive, real-time or batch. The scope of this thesis is restricted to a heterogeneous computing environment. This thesis presents a methodology, based upon discrete event simulation, for finding the optimal point at which to schedule jobs entering a system. It also demonstrates how this solution can be integrated into a heterogeneous computing architecture, such as the Management System for Heterogeneous Networks (MSHN)[1].

A. BACKGROUND

The DARPA QUORUM program seeks to develop a broad range of resource management, networking, and data management technologies to support heterogeneous, distributed, real-time applications. As part of the QUORUM program, NPS is designing and implementing the Management System for Heterogeneous Networks (MSHN). This project's goal is to provide an infrastructure to support the execution of concurrent, dynamically changing C4I, radar processing and weapon control applications in an environment consisting of multiple shared heterogeneous resources. MSHN must account for dynamically changing priorities as well as resource availability. Two of the many areas being researched for the MSHN system include examining scheduling

[1] Pronounced "mission"

policies for heterogeneous environments and determining when to cause a job to adapt in response to a changing situation.

In a networked environment, computing jobs are assigned by some mechanism to a machine for execution. Currently, most networks rely on the user to perform this assignment. One of the goals of MSHN is to efficiently and automatically determine the "best" placement of jobs on machines in order to optimize the QoS achieved by the jobs.

QoS can have different meanings depending upon context and perspective. For example, from a user's perspective, QoS may be measured in terms of quickness or accuracy, while a (computer) application's perspective may be based upon quantities such as cycles or frame rate. Additionally, an entire system such as MSHN generally perceives QoS differently from either the user or the application. The QoS delivered by MSHN needs to be measured in terms of balancing the dictates of the system against satisfying the needs of the applications, which are in turn satisfying the needs of the originating users. Optimizing the QoS achieved by applications involves both job scheduling and resource management, each of which carries its own computational overhead. As the degree of dependency grows between jobs, this computational overhead becomes increasingly significant. The problem being studied in this thesis centers on the group of jobs waiting to be scheduled. The focus of investigation for this thesis research is how the overall system QoS changes as the size of the group of jobs waiting to be scheduled changes.

There are two important issues relating to this focus. The first is the effectiveness of the schedule produced. Many computer scheduling algorithms aim to utilize the available resources in the most efficient manner, while still satisfying QoS requirements. However, this research will not be examining the performance of scheduling heuristics or algorithms as there has already been much research done in this area (see references [ARMS98], [BRAU98] and [BRAU99]). The second is the performance of the scheduling mechanism itself. The scheduling process requires overhead that utilizes some resources, particularly if there are many dependencies between jobs. Additionally, this time-consuming scheduling causes resources to lie idle awaiting the assignment of jobs. Of course, good scheduling becomes most important when the system is being heavily used. An effective resource management system must determine when to

compute a schedule and if it is advisable to accumulate additional requests before scheduling. This thesis concentrates on these issues.

B. PROBLEM STATEMENT

The intent of MSHN is to provide middleware that extends the functionality of individual operating systems (OSs) in an HC environment in such a way that the extended OS provides the required and requested QoS. Good application scheduling is a necessary component to achieving this goal. This research attempts to answer two questions related to effective scheduling in a heterogeneous computing environment. The first focuses on finding the optimal point at which to submit a group of jobs waiting to be scheduled. Intuitively, the larger the number of jobs we can submit at any given time, the better schedule we can produce. However, if we wait too long to collect those jobs, the queued jobs will be delayed. Thus, the intent of this research is to determine the cost tradeoff relationship between the potential performance improvement obtained from submitting a larger pool of jobs to the scheduler, versus the potential performance loss due to jobs that must wait for that particular pool size to collect. This relationship will be used to determine the optimal size of the job pool, that is, the point at which to submit jobs to the scheduler.

The second question is to determine whether there is a point beyond which the relative benefit of submitting a group to the scheduler substantially decreases as the size of the group increases. For instance, when given the situation where jobs are arriving at a rate more rapid than the rate at which jobs are completed, the size of the group of jobs to be submitted could become very large. In this case, the time to execute the scheduling algorithm may become prohibitively long in comparison to the benefit achieved from the resultant improved schedule. Once this point has been reached, it would be better to simply schedule a smaller group of jobs, accept the lesser schedule, and forward the jobs to the available resources.

The quantitative answers to these questions can then be used to help ensure delivery of the requested QoS, and so support the QoS goals of jobs executing under MSHN.

C. GOAL

It is beyond the scope of this thesis to answer both of the questions posed above for all possible environments. However, as part of this thesis research, this author developed a computer simulation framework that uses the Monte Carlo Method along with Discrete Event Simulation (DES) that can later be used to answer these questions in a more general sense. The framework permits the user to vary the input distributions that are fed to the computer simulation of a heterogeneous computer resource system. The resulting statistical output can then be analyzed to determine a solution to the problems stated previously under many different conditions. Additionally, this thesis answers the stated problems for a specific subset of input and using a scheduler whose goal is to minimize the time at which the last job, in a series of jobs, completes. This research, in contrast, is attempting to minimize the average time a job spends in the system. Finally, the thesis research will demonstrate how the methodology and results of the experiment can be generalized for any defined QoS measure.

D. THESIS ORGANIZATION

This thesis is organized as follows: Chapter II provides a brief overview of the MSHN architecture and puts this research into perspective. Chapter III discusses simulation models and methods. It introduces key concepts and provides background on Discrete Event Simulation, Monte Carlo Methods, and Queueing Theory. Chapter IV describes the specific simulation model built for this research and provides a brief synopsis of the Silk programming environment using JAVA. Chapter V details the experimental data obtained from the simulation and describes the results obtained from those experiments. Chapter VI provides the statistical analysis of the data detailed in the previous chapter, describes the techniques used in the analysis, and summarizes the conclusions. The final chapter summarizes the conclusions of this thesis and describes future work.

II. MANAGEMENT SYSTEM FOR HETEROGENEOUS NETWORKS

A. INTRODUCTION

The goal of this chapter is to summarize the high level goals and architecture of the Management System for Heterogeneous Networks (MSHN) and, particularly to explain how this thesis relates to that project. Section B explains the purpose of MSHN. Section C describes the architecture of MSHN and how its components interact. A description of how this thesis research fits into the overall goal of MSHN is provided in Section D. Section E summarizes how this thesis will benefit the MSHN project. Those already familiar with MSHN can safely skip sections B and C.

B. PURPOSE

Nearly as quickly as the computational power of personal computers has increased, the price for those machines has decreased. Additionally, the performance of the networks connecting computers has rapidly improved. These advancements, along with decreasing operational budgets, have driven both governmental agencies and private corporations to move away from large, central mainframe computer systems towards satisfying their computational needs via more distributed systems. These distributed systems consist of many individual computer resources connected by at least one network. These networks can span great distances and include resources or machines of varying types. There are essentially five major advantages provided by distributed systems: resource sharing where hardware and software resources can be shared by multiple computers; enhanced performance provided by the fact that many tasks can be concurrently executed by different computers; improved reliability through the replication of data providing fault tolerance; improved availability because some elements of the system can fail without affecting the accessibility of the rest of the system; and modular expandability where new hardware and software can be added without affecting the rest of the system [SING94]. The increased use of distributed computer systems has resulted in the need to develop methods and systems for effectively managing large heterogeneous networks of computers in order to deliver good QoS

performance to all users. To respond to this requirement, a resource management system (RMS) named MSHN is being developed as part of the Defense Advanced Research Projects Agency's (DARPA's) QUORUM program. The goal of MSHN is to provide a computing environment that delivers, "whenever possible, the required quality of service (QoS) to individual processes that are contending for the same set of distributed, heterogeneous resources." [HENS99] In other words, when given a set of jobs, MSHN will automatically determine the "best" placement of jobs on resources in order to optimize the required QoS as specified by the user.

MSHN evolved from SmartNet, which was a scheduling framework developed by the Heterogeneous Computing Team at the US Naval Command, Control, and Ocean Surveillance Center's (NCCOSC's) Research, Development, Test and Evaluation (RTDE) Division in San Diego, California. SmartNet's design implemented scheduling algorithms for minimizing the time at which the last job, of a set of computationally intensive jobs, finished on a suite of heterogeneous computing resources. It also provided the necessary information for these algorithms to make wise decisions [KIDD96]. SmartNet treated the set of available compute resources as one virtual heterogeneous machine (VHM), achieving superior performance by mapping applications to resources based upon its knowledge of the VHM and the characteristics of the jobs it executed. A more detailed description of SmartNet is available elsewhere (see references [KIDD96], [JANA96], and [ARMS97]).

However, the expanded goal of MSHN differs from SmartNet in three major ways. First, "MSHN needs to consider that the overhead of jobs sharing resources, such as networks and file servers, can have significant impact upon mapping and scheduling decisions." [HENS99] Second, MSHN needs to deliver good QoS simultaneously to users who may be executing either Input/Output intensive, compute intensive or real-time jobs. Lastly, MSHN must support idempotent applications that can produce results using one of a variety of algorithms, or can exist in several different versions or forms. These applications are called adaptive or adaptation-aware applications. [HENS99]

As an RMS, MSHN must be capable of providing the user with a location-transparent view of its available set of heterogeneous resources, while providing improved performance. To accomplish this, the MSHN RMS is not stand-alone software,

but an integrated system architecture incorporating a variety of distributed components that strive to attain the maximum benefits from available resources. This architecture is detailed in Section C.

C. ARCHITECTURE

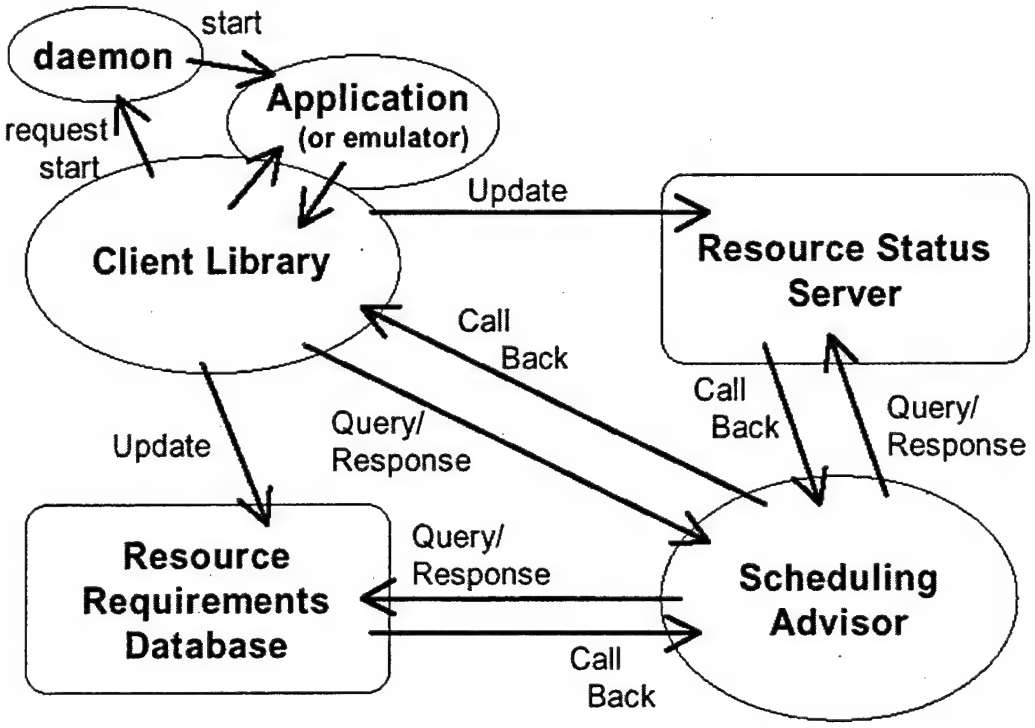


Figure 1. MSHN's conceptual architecture, April 1999, from [HENS99].

The MSHN architecture is continually evolving. Hence, the following description is based upon the architecture design as of April 1999. While the final MSHN architecture may differ slightly from this description, the core components and concepts are not expected to change. Figure 1 shows the conceptual architecture for MSHN and demonstrates how these various components interact.

The MSHN RMS consists of a client-server architecture that will be comprised of the following core components:

- the Client Library (CL),
- the Scheduling Advisor (SA),
- the Resource Requirements Database (RRD),

- the Resource Status Server (RSS),
- the MSHN Daemon, and
- the Application Emulator (AE).

The following is an overview of the entire architecture and a high level description of each of the components.

The entrance to the MSHN RMS is the Client Library. The CL permits MSHN to function without requiring the user to explicitly log into the MSHN RMS. It does this by transparently intercepting calls to system libraries and diverting those calls to the CL. By pre- and post-processing these system calls, the CL is capable of determining the resources used by an application, sending this data to the RRD, updating the RSS while the process continues to execute, and sending a request to the SA to determine where best to run the application in order to support the application's QoS requirements. More precisely, when the CL receives a request to launch a new application, it first checks the request against a list of applications managed by MSHN. If the request is on that list, the CL passes the request to the SA, otherwise, the requested application is simply passed to the local operating system. When the request is sent to the SA, it includes the QoS requirements defined by the user.

It is then the SA's job to determine which set of resources the requested application should use. To make this determination, the SA consults the RRD, which maintains information about the resources that are required to execute any particular application. The SA also questions the RSS to obtain information concerning the current availability of resources. Using the information from these two sources and the optimization criteria derived from the user-requested QoS, the SA then decides where to execute the process and returns this information to the requesting CL.

The CL then asks the Daemon located at the selected computer to execute the application. Whenever a computer is added to the system, a MSHN Daemon is started on that computer. The purpose of the Daemon is to receive requests from remote CL's and to start processes on the local computer on behalf of the remote CL. Daemons are also used to start specially designed AE's that are used to determine resource status information. Once applications have been started, the CL is also capable of updating the

RSS and RRD with the current status of the resources and the requirements of the current process. Concurrently, the SA establishes callbacks with the CL, the RSS, and the RRD so that it can be notified in the event that either the status of the resources has significantly changed or the resource requirements have changed from what was initially reported. In either case, the CL is notified by the SA and requests Daemons on other remote machines to launch preferred versions of adaptive applications in accordance with the user's requested QoS.

An additional component of MSHN is the Visualizer that permits examination of the current states of the other MSHN core components. The Visualizer captures significant events within and between the core components for both real-time and post-mortem analysis. More detailed information about MSHN can be found at www.mshn.org and in previously published references (see references [HENS99], [SCHN98] and [PORT99]).

D. THESIS APPLICABILITY

The results of this thesis research will be applied to the functionality of the SA within MSHN. The primary responsibility of the SA is to determine the best assignment of resources to a set of applications. To function, the SA depends upon the RRD and the RSS to identify an operating point that attempts to optimize a global measure[2]. It responds to assignment requests from the CL and when necessary requests application adaptations via the CL. Lastly, the SA sends status updates to the MSHN Visualizer.

This research focuses upon identifying this optimal operating point's dependency upon the quality of the schedule produced. In other words, as the SA receives requests from the various MSHN CLs, should it schedule each request individually or should it wait until a certain number are received before beginning its processing. It was mentioned in Chapter I that when more information is available about the jobs to be scheduled, the better the resultant schedule will be. Therefore, we must determine the

[2] Note that throughout the thesis we use the phrase "attempts to optimize" rather than "optimized" because the scheduling problem that is examined is known to be NP-complete and the number of jobs to be scheduled is typically too large to use an exponentially complex algorithm.

best point at which to initiate the scheduling process with respect to the number of jobs submitted.

E. SUMMARY

This chapter described the MSHN resource management system currently under development. MSHN's purpose is to manage jobs, some of which may be adaptive, in a heterogeneous environment, with the goal of delivering good quality of service. Additionally, its architecture was outlined as a collection of core elements that, when interacting, provided the desired result. Lastly, a brief description was provided as to how this research will benefit MSHN in general, and the Scheduling Advisor in particular. The identification of the optimal point at which to conduct the scheduling process will improve the performance of the Scheduling Advisor and, hence, MSHN. The next chapter will introduce the concepts of simulation and queueing theory that will be used to outline the detailed problem definition and experimentation model in Chapter IV.

III. DISCRETE EVENT SIMULATION AND MONTE CARLO METHODS

A. INTRODUCTION

This chapter explains what discrete event simulation is and discusses other related topics. Section B discusses simulation in general and explains why computer simulation has become such a useful tool. Section C describes discrete event simulation in detail. Section D provides insight into the Monte Carlo Method and how it is distinct from Monte Carlo Simulation. Section E explains the concept of random variates. A general introduction to queueing theory is presented in Section F. Section G provides a summary and concluding remarks.

B. BACKGROUND

The need to predict the operation of real-world processes or systems before implementing expensive projects has led many researchers to attempt to simulate these systems using computers. A later example will illustrate why simulation is so powerful. First, we need to describe the types of systems that can be analyzed using simulation techniques. Systems can take many forms, the most common being the following:

- a physical entity such as an actual machine or an industrial facility,
- a non-physical entity such as an environmental phenomenon or human interaction, and
- a process of events involving both physical and non-physical entities such as an automobile manufacturing plant incorporating both machines and humans in the production process.

Our example will be of the last type.

Thus, a system is a collection of entities, such as people or machines, which act and interact collectively toward the accomplishment of some goal. The state of the system is the collection of variables needed to describe the system at a particular time [LAW91].

As an example of a complex system, consider a communication and information system used by a large military organization in combat. This system involves both physical entities, such as its radios, telephones and computers, and the non-physical, human interaction based processes required for the system to operate. There are several methods available to study any system. Figure 2 maps the different ways that a system could be studied.

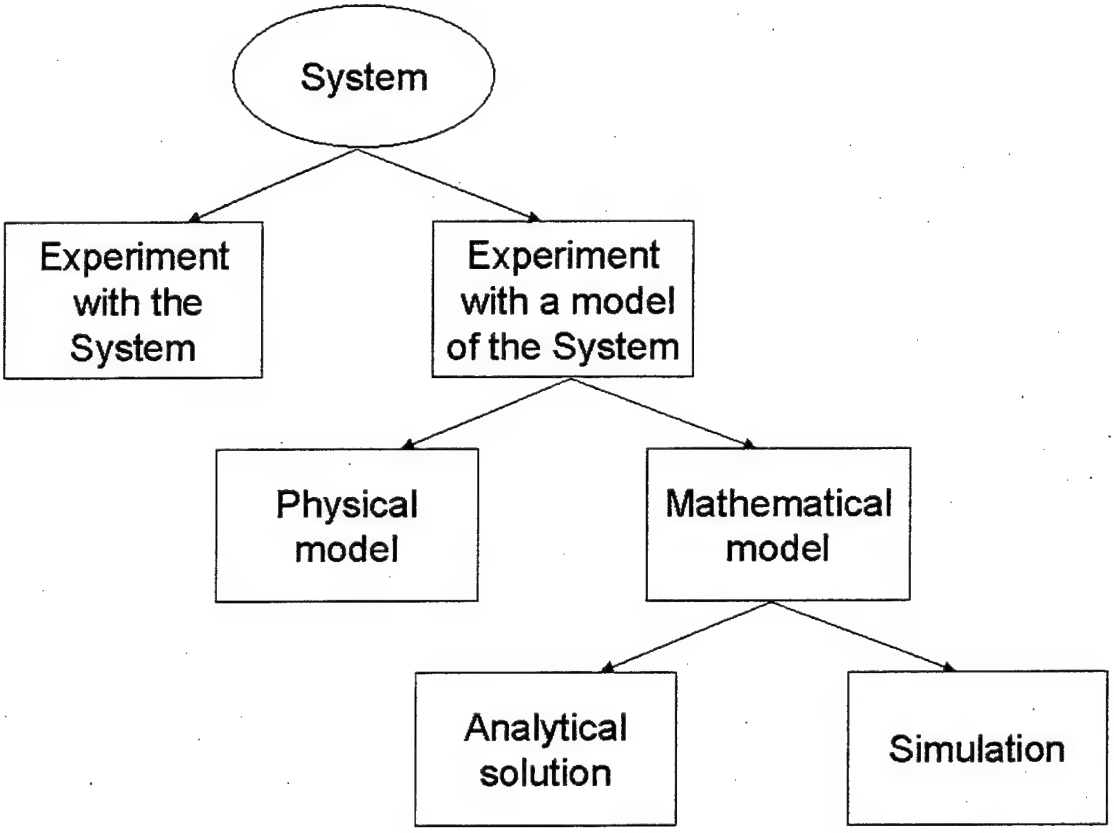


Figure 2. Ways to study a system, from [LAW91].

The most obvious method is to study the actual system. In our example, this would involve moving all of the radio, microwave, and satellite transmission equipment, the telephone switching units, computer devices, and related ancillary user equipment from a storage or "garrison" site to an operating or "field" site. Additionally, all of the personnel who normally use and operate the equipment would be required to abandon their garrison duties and move into the field to operate and test this information system. It should be obvious that this method is very time consuming and costly. It would involve: actually moving the people and equipment to a location; building, operating and

maintaining the system; altering the system as determined from the study; and finally returning the people and equipment to their initial state. Thus, experimenting with the actual system is frequently infeasible. For this reason, it is often necessary to build a model or representation of the system and study the model instead.

A model of the system can be built either physically or mathematically. Physical models are rarely used for systems analysis due to their required complexity. For example, building a physical model of our information system would be nearly as infeasible as using the actual system. A high fidelity physical model of this system would involve reproducing each element of the system in a reduced size and then attempting to duplicate the human-machine interaction of the system. The difficulty in making such a model accurate is obvious and would render a study of the model as a surrogate to the actual system as costly, if not more so, than experimenting with the actual system.

An alternative to a physical model is a mathematical model. Such a model represents a system in terms of logical and quantitative relationships that can be easily manipulated or altered to demonstrate how the model, and hence the system, reacts. In our communication example, the mean time between equipment failures could be mathematically modeled as a deterministic amount of time generated from a probability distribution. The size of communication messages and the length of time it takes to send a transmission from one point to another could be similarly modeled using an average determined from historical data. Overall, the cost of building and experimenting with a mathematical model can be an order of magnitude less than that of a physical model or of experimenting with the actual system. Additionally, the modeler can easily refine and modify the fidelity of various parts of the system model as the importance of various requirements change.

Once the decision to use a mathematical model has been made, the designer must determine if an exact analytical solution can be found or if using a simulation method would be more appropriate. In cases where the model is simple enough, it may be possible to work with its quantities and relationships in such a fashion that a closed-form analytic solution can be found. However, the combined interrelationships of critical elements within mathematical models can become extraordinarily complex. Once

systems and their corresponding mathematical models reach this level of complexity, there is little choice but to study the model using simulation. Said simply, simulation is numerically studying the model for the inputs in question with the goal of determining how these inputs affect the output measures. As an example, we consider modeling the complex information system described earlier.

One part of the information system is the telephone switching unit. To model a single switch, we must mathematically represent items like the mean time between failure of the processor, the memory unit, and the generator supplying power to the switch. Also, rate of failure of the operator, arrival rate of requests, and down time due to maintenance must be modeled. It is obvious that there are numerous details that must be considered when modeling a single telephone switch, and there may be a total of fifteen switches in use, each of a different type. Additionally, the switch is only a very small part of the entire information system being studied. The complexity of modeling just one switch may be great in and of itself, but when these are combined with the rest of the system, it is apparent that a simple analytic solution is not possible. Further, additional complex pieces of the information system need to be included in the model such as battery consumption for the radios, fuel consumption for the field generators and the human factor relating to the users and operators.

In every environment where people are working under stressful conditions, accidents and injuries occur. The rates of these incidents must be modeled. Consider a young satellite equipment operator who has had little sleep in the past 48 hours. The operator is assigned the task of erecting a new satellite antenna to replace one that is not functioning, and accidentally forgets to remove a retaining pin before engaging the hydraulic lift. This accident causes the lifting arm to shear, which causes the dish to fall and collapse. Additionally, the jarring motion of the accident causes the operator to fall from the equipment, thus breaking a leg. We see that such scenarios, when modeled with great fidelity, become mathematically very complex, eliminating the possibility of a closed-form solution.

Simplification is an important aspect of modeling. Simplification is the process of reducing or removing complex elements from a model while only minimally reducing its accuracy or usefulness. Taking the telephone switch example from above, if the

switch only fails once every three months and the average usage period is less than three months, then the modeler can assume that the failure rate is zero, thus simplifying the model. However, careful consideration must be taken when making simplifying assumptions. If the switch actually failed once every four hours, that factor would need to be included in the model.

A simulation, usually conducted on a computer, also uses a mathematical model. In these cases, the computational power of the computer is used to simulate the complex mathematical quantities and relationships of the model. In these simulations, it is fairly easy to increase the fidelity of one aspect of the model while decreasing the fidelity of others. This aspect is important as various factors within the model increase and decrease in significance. For example, during one examination of the information system, the rate of failure and energy consumption of the telephone switch may be significant, while at another time it may be sufficient to only consider the number of calls completed per hour. This simplification would reduce the complexity of the model and may make it easier to evaluate. These changes to the model are often more easily made in a computer simulation model than in an analytical model.

Given that we have a mathematical model that we wish to study using the simulation method, we must ascertain how to accomplish this task. To begin, the type of simulation must be determined from the classification factors below.

- **Static versus Dynamic Models.** A static model is a depiction of a system at a particular instant in time, or a model where time is not a variable. The system state in a dynamic model changes over time making the model evolutionary.
- **Deterministic versus Stochastic (Probabilistic) Models.** A model where the results can be predicted with certainty, given a particular set of inputs, is called deterministic. A stochastic model gives, for each execution, a different result for the same set of input parameters, or it contains probabilistic components that causes the output to be considered only an instantiation of a single case. Many repetitions are needed in order to obtain a statistically significant estimate of the true model characteristics.

- **Continuous versus Discrete Models.** A model where the system state is defined at all times is called a continuous model. If the system state is only defined for particular, finite points in time, then the model is discrete.

The simulation designed for this thesis is discrete, dynamic, and stochastic in nature. This type of simulation is commonly termed a discrete event simulation.

C. DISCRETE EVENT SIMULATION

Discrete event simulation (DES) models a system as it evolves over time by representing instantaneous changes in the state variables at separate points in time. These points in time are the ones at which events occur where an event is defined as "an instantaneous occurrence that may change the state of the system." [LAW91] Events occur at different times and are stamped with the simulation time at which they occurred. The system state is defined by system specific state variables that change as the simulation progresses, and thus, describes the system's condition at any given point in time. Current events generate additional events to occur at some future point in time. These future events are stored in an "event queue" where they remain until the simulation clock advances to the time at which the events are to happen. Events in the event queue are normally stored in order according to the simulation time at which they are to occur. As a discrete event simulation progresses, events are removed from the event queue and processed. When events are removed, the simulation time is correspondingly advanced to the time stamped on the next current event. This progression of simulated time is one reason why simulation time and elapsed real-time are usually different in a discrete event simulation.

Characteristically, DES models require three different types of variables:

- **System State:** model/system dependent variables used to describe the system at particular points in time.
- **Simulation Clock:** a global variable representing simulated time.
- **Statistical Counters:** variables used to track repetitions of certain events and to store statistical data about system performance.

In DES models, the advancement of time can be a difficult concept to understand. There are generally two approaches for advancing simulated time:

- **Next-event time advance (event driven):** time is advanced to the time of occurrence of the next future event, at which point the state of the system is updated to account for the occurrence of that event. Most major simulation tools and people coding in general purpose languages use this approach [LAW91].
- **Fixed-increment time advance (unit time):** time is advanced by fixed increments and checks are conducted to determine whether any events were scheduled to occur during the previous time interval. If events were scheduled to occur during this interval, these events are considered to occur at the end of the interval and statistical counters are updated accordingly. This approach is not commonly used in computer simulations [JAIN91].

Although simulation has been applied to many varied types of real-world systems, DES models all share common components and a universal structure [JAIN91]. The use of these common components promotes the coding, debugging, and future changing of a simulation model's computer program. While most simulation languages and tools provide many of these components, when using a general-purpose language to code a simulation, the modeler must develop them. The components are as follows:

- **Event list:** a list, commonly implemented using a queue, of events and the times when they should occur.
- **Initialization routine:** a subprogram that sets the initial system state, receives values from the user for input parameters, and sets the simulation time to zero.
- **Timing routine:** a subprogram that determines the next event from the event list and advances the simulation clock to the time when that event is to occur. Often referred to as the event scheduler.
- **Event routine:** each event is simulated by its corresponding routine that updates the system state variables and schedules other events.
- **Library routines:** a set of subprograms that generate random observations from probability distributions.

- **Report generator:** output routines that compute results from the statistical counters and produce a report when the simulation is completed.
- **Main program:** a subprogram that brings all of the routines together. It initializes the simulation, executes various iterations, and upon finalization, calls the report generator.

The logical relationship among these components and the flow of control of a next-event time advance discrete model is illustrated in Figure 3.

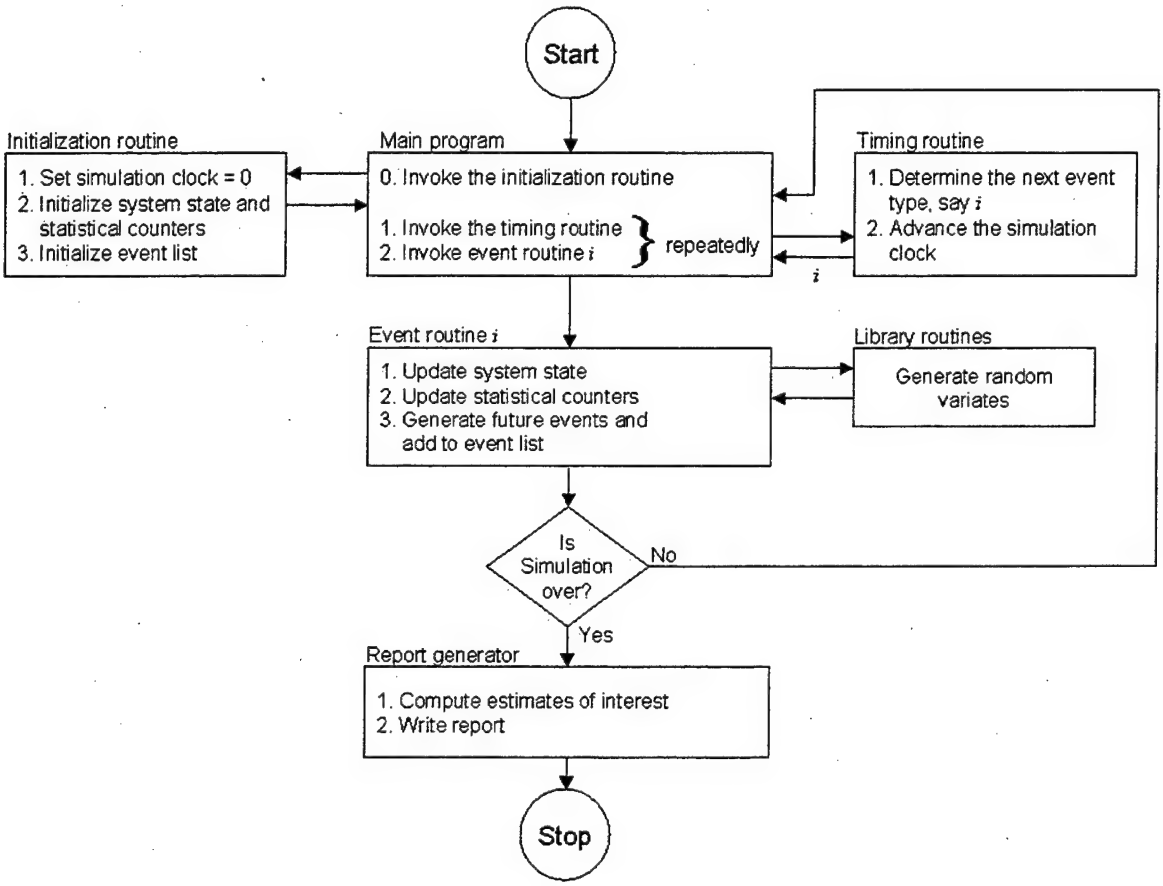


Figure 3. Flow of control for the next-event time advance approach, from [LAW91].

Once the type of simulation is determined, a choice must be made as to the method of experimentation. The method used for this thesis research is the Monte Carlo method, an approach that is often confused with Monte Carlo simulation. A detailed description of this approach and why it is often misnamed Monte Carlo simulation can be found in Section D.

D. MONTE CARLO METHOD

First, we present a review of items presented previously. The very nature of a simulation is to provide statistical estimates of numerical data about the performance of a system being modeled. When simulation experiments are conducted over time, we have a stochastic simulation that includes sampling stochastic variates from a probability distribution. A stochastic simulation is a simulation involving a stochastic process, a process with a sequence of states whose development is determined by randomly occurring events. Because sampling from a particular distribution involves the use of random numbers, stochastic simulation is sometimes loosely referred to as Monte Carlo simulation. However, a more precise definition of Monte Carlo simulation is "a scheme employing random numbers, that is U(0,1) random variates, used to solve stochastic or deterministic problems where passage of time plays no substantive role." [3] [LAW91]

It can also be stated that a Monte Carlo simulation is a static simulation (i.e., one without a time axis) used to model probabilistic events that do not change characteristics with time [JAIN91]. Thus, Monte Carlo simulation is useful when evaluating nonprobabilistic expressions with probabilistic methods. For example, consider evaluating the following integral:

$$I = \int_0^2 e^{-x^2} dx$$

One method to evaluate this integral is to generate uniformly distributed random numbers x and for each number compute a function y as follows:

$$\text{density function} \quad f(x) = \frac{1}{2} \quad \text{iff } 0 \le x \le 2$$

$$x \sim \text{U}(0,2)$$

[3] A U(0,1) random variate is obtained from a random number generator that produces a variate U that is uniformly distributed between 0 and 1.

The expected value of y is then:

$$E(y) = \int_0^2 2e^{-x^2} f(x)dx$$
$$= \int_0^2 2e^{-x^2} \frac{1}{2} dx$$
$$= \int_0^2 e^{-x^2} dx$$
$$= I$$

Thus, generating uniformly distributed random numbers x_i, computing y_i, and then averaging can evaluate the integral.

$$x_i \sim U(0,2)$$
$$y_i = 2e^{-x^2}$$
$$I = E(y) = \frac{1}{n} \sum_{i=1}^{n} y_i$$

However, for our research, we are interested in the dynamic rather than static aspects of the Monte Carlo method. The above example demonstrates Monte Carlo simulation, but Monte Carlo methods can be used for a much broader range of experiments.

Monte Carlo methods can be defined as "the branch of experimental mathematics concerned with random numbers where the approach is to observe random numbers, chosen to directly simulate the physical stochastic process of the problem, and to estimate the desired solution from the behavior of the generated random variates." [HAMM64] Thus, Monte Carlo methods can be used in any experiment where some input parameters are random variates and the desired output is a calculated estimate obtained by using statistical methods on the results of numerous trials of an experiment. The general flow of a simulation using Monte Carlo methods is shown in Figure 4.

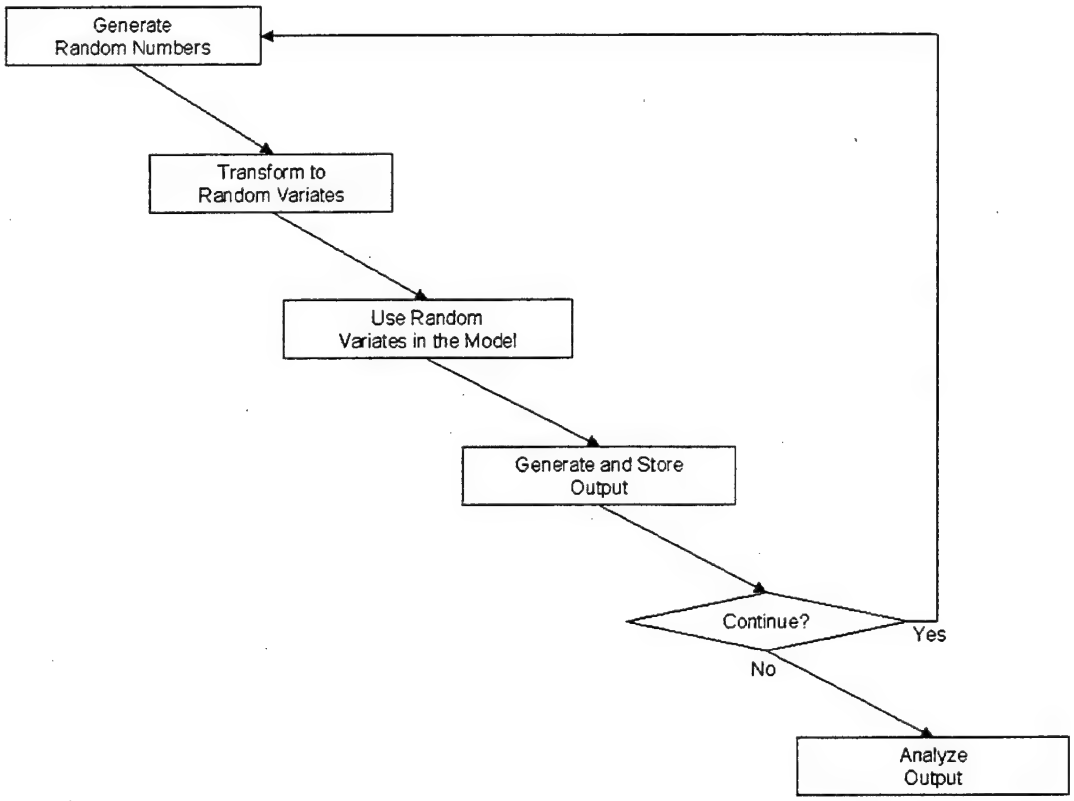


Figure 4. Flow diagram for Monte Carlo method.

Although references to Monte Carlo type experimentation can be found much earlier, the actual name and development of the process dates back to about 1944 where "Monte Carlo" was the code word for the secret work on the atomic bomb during World War II at Los Alamos. This work was conducted by von Neumann and Ulam and involved direct simulation of probabilistic problems concerned with random neutron diffusion in fissile material [RUBI81].

In conclusion, simulation is a technique for performing sampling experiments on a model of a system. When these experiments are conducted over time, we have a stochastic simulation involving sampling stochastic variates from a probability distribution. Because sampling from a distribution involves the use of random numbers, stochastic simulation is sometimes also erroneously referred to as Monte Carlo simulation. A more precise denotation for this type of simulation would be stochastic simulation using Monte Carlo methods. Therefore, this research uses discrete event simulation and the Monte Carlo method of experimentation.

E. RANDOM VARIATES

The nature of discrete event simulation requires it to include the random elements of the modeled system through the integration of stochastic processes [ARMS97]. To illustrate, we again refer to the telephone switch example outlined previously in this chapter. While the process of automatically connecting telephone calls is normally a routine and consistent operation, there are still unavoidable random elements in the system. For example, consider the possibility of a processor failing in the system. During its operation, data on the mean time between failures for the telephone switch processor is collected. This data is then statistically analyzed, the mean and variance determined, and a probability distribution fitted to the processor failure rate. A failure of the processor can then be simulated as a random occurrence governed by the distribution fitted to the observed data. Consequently, this stochastic simulation could be used to demonstrate the effect of a switch with a given reliability on the overall performance of the larger communication network.

An important aspect of the above simulation is the efficient and correct production of random variates. A random variate is a random observation generated from a probability distribution [LAW91]. The Gaussian (Normal) distribution is a good example to use to explain the concept of random variates. Consider a set of 100,000 random variates sampled from a Gaussian distribution with a mean of 100 and a standard deviation of 15. (See Figure 5). The *x*-axis values represent random variates with the frequency of those variates plotted along the *y*-axis. The Gaussian curve shows that there are more random variates near the mean and less as we move away from the mean.

The technique used to generate random variates depends upon the particular distribution that we wish to sample, but every method relies upon a reliable source of independent, identically distributed (IID) random variates uniformly distributed over the interval (0,1) [LAW91]. For this reason, it is essential that a statistically reliable U(0,1) random number generator be available. Fortunately, most computer simulation tools today have such convenient and accurate random number generators. Additionally, most simulation tools also include convenient methods for generating random variates. However, if they do not, the approaches described next for generating variates can be used.

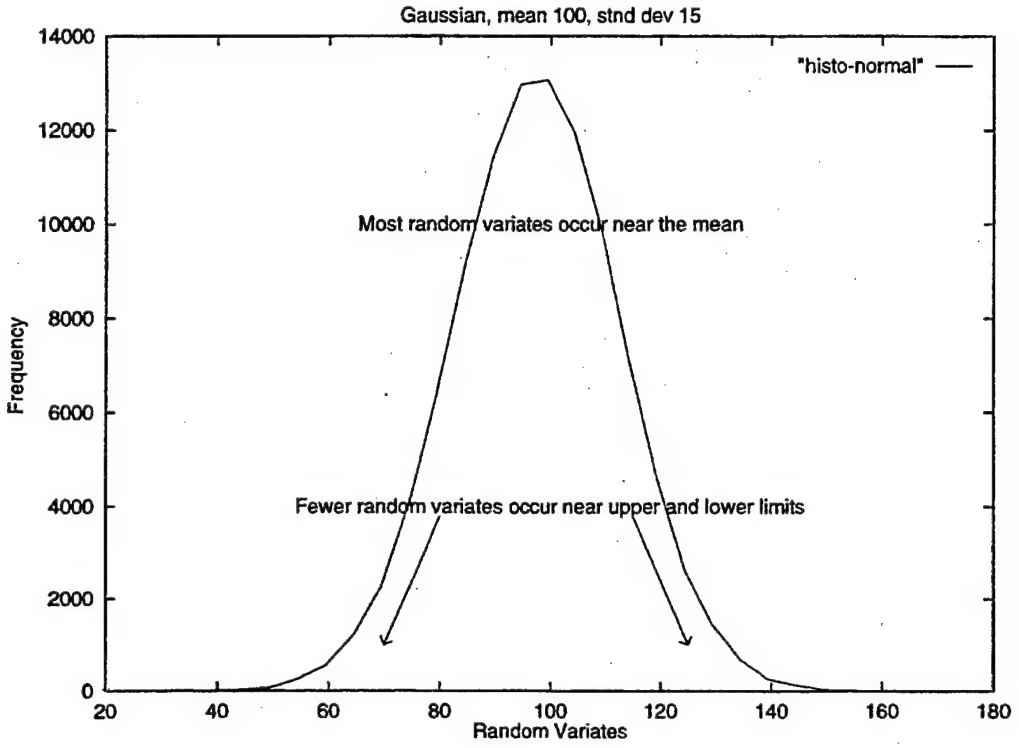


Figure 5. Gaussian distribution, mean 100, standard deviation 15, from [ARMS97].

- **Inverse Transform.** This method is used to generate random variates whose distribution function F is continuous and increasing when $0 < F(x) < 1$. The algorithm for generating random variate X is then to generate $U \sim U(0,1)$ and then return $X = F^{-1}(U)$. Since $0 \leq U \leq 1$ and the range of F is [0,1], F^{-1} will always be defined. [LAW91]

- **Composition.** This technique applies when the distribution function F can best be expressed as a combination of other distribution functions. When F can be expressed as a convex combination of other distribution functions F_1, F_2,..., F_n it may be easier to sample from the individual F_j's than from the original F. The combination algorithm generates a positive random integer J such that $P(J=j) = p_j$ for $j = 1, 2, \ldots, n$ and then return X with distribution function F_J.

- **Convolution.** This method is best when the desired random variable X can be expressed as a sum of other IID random variables that can be more readily

generated than the direct generation of X. This method is fundamentally different from the method of composition because in composition we assume that the distribution function of X is a (weighted) sum of other distribution functions, whereas the assumption behind convolution is that the random variable X can be represented as a sum of other random variables. The algorithm for generating the random variate X is quite intuitive because we first generate the independent, identically distributed variables $Y_1, Y_2,\ldots,Y_m$ each with distribution function G and then return $X = Y_1 + Y_2 + \ldots + Y_m$. [LAW91]

- **Acceptance-Rejection.** This technique is less direct in its approach than the aforementioned methods and is useful when the direct methods fail or are too costly. This approach requires a function t that majorizes[4] the density function f. The general algorithm involves generating a random variate Y having density r (hopefully easily and quickly), and generating $U \sim U(0,1)$, independent of Y. If $U \leq \frac{f(Y)}{t(Y)}$, return $X = Y$. Otherwise it generates a new value and tests it similarly.

The appropriate method to use is dependent upon the distribution from which we desire to draw the random variate. The ease and reliability with which random variates can be generated for that distribution must also be considered. (The reliability is based upon the fact that the algorithm used should produce random variates statistically equivalent to the desired distribution.) [LAW91]

For distributions of known types, most simulation tool kits provide readily available, efficient, and accurate random variate generators. However, when necessary, the inverse transform method is the easiest to implement [ARMS97]. This method is easiest because the random variates are generated from the inverse of the distribution function F, provided the inverse function F^{-1} is defined. For example, the Gaussian distribution function cannot be inverted because a closed form version for F^{-1} does not exist. While there are other numerical methods available when F^{-1} has no closed form,

[4] $t(x) \geq f(x)$ for all x.

the inverse transform method may not be the most computationally efficient. For the cases when the distribution function can be expressed as a sum of other distribution functions or the random variate is a sum of other random variates, then the composition or convolution methods should be used, respectively. Finally, if no other more efficient method can be found and a majorizing function exists, the acceptance-rejection method should be used.

F. QUEUEING THEORY

The amount of information and literature on queueing theory is immense. This section presents a very general synopsis of queueing theory. This topic is important because "queueing theory is a key analytical modeling technique used for computer systems performance analysis." [JAIN91] To aid in the understanding of queueing theory, this section provides a basic knowledge of queueing notation and some background on single-queue systems. For systems with multiple queues, as in this thesis, simulation is often required. Queueing theory assists in determining the time that jobs spend in various queues throughout a system. These times can then be combined to predict the system response time, which is the total time that a job spends in the system.

To begin, we introduce queueing notation. It is based upon a shorthand called the Kendall notation, and has the form A/S/*m*/B/K/SD where the letters correspond to the following parameters:

- **A: interarrival time distribution** – The times between job arrivals are often assumed to be a sequence of IID random variables. The most common arrival process has exponentially distributed interarrival times, and is often referred to as the Poisson arrival process.
- **S: service time distribution** – The time each job spends at a resource is called the service time. The service times are also usually assumed to be IID random variables. For simple queueing models, the exponential distribution is commonly used.
- ***m*: number of servers** – Identical servers are commonly grouped together and considered part of the same queueing system. When the servers are not all

identical, they are grouped into disjoint sets, each set containing identical servers, with individual queues serving each group.

- **B: system capacity (number of buffers)** – The system capacity is the maximum number of jobs that can be serviced by the system. This number is usually finite and includes both those waiting for service and those already in the system.
- **K: population size** – The total number of jobs that could ever possibly enter the system is called the population size. This number is usually finite.
- **SD: service discipline** – Service discipline is the order in which jobs are served. The most common discipline is First Come First Served (FCFS).

Additionally, the service and interarrival time distributions are ordinarily referred to by a one-letter symbol. The most common symbols are:

- M - Exponential[5]
- E_k – Erlang with parameter k
- H_k – Hyperexponential with parameter k
- D – Deterministic
- G – General

A deterministic distribution implies that the times are constant and there is no variance, while a general distribution means that the results are valid for all distributions because the distribution is not designated. [JAIN 91]

There are several important variables used for analysis that are common to all single queueing systems. These variables are:

- τ = the time between successive arrivals, or interarrival time.
- λ = the mean arrival rate, commonly $\lambda = 1/E[\tau]$.
- s = the service time per job.
- μ = the mean service rate per server, commonly $1/E[s]$.
- n = the number of jobs in the system.
- n_q = the number of jobs waiting for service.
- n_s = the number of jobs receiving service.

[5] The letter M is used to indicate the Markov, or memoryless, property of the exponential distribution.

- r = the total time in the system or response time.
- w = waiting time or the time between arrival time and the instance that service begins.

Except for λ and μ, the above variables are random variables. [JAIN91]

A vital theorem used in queueing theory is Little's Law. This law relates the number of jobs in the system to the mean response time as follows:

Mean number in the system = arrival rate X mean response time

As long as the number of jobs entering the system is equal to those completing service, this law applies. In other words, no new jobs can be created within the system and no jobs can be lost forever by the system. This law may be applied to the system as a whole or to individual parts of the system.

To conclude this section, Table 1 is provided as a synopsis of how to analyze a standard, single-server, system represented by the common M/M/1 queue.

1. Parameters:
   λ = arrival rate in jobs per unit time
   μ = service rate in jobs per unit time
2. Traffic intensity: $\rho = \lambda/\mu$
3. Stability condition: Traffic intensity ρ must be less than 1.
4. Probability of zero jobs in the system: $\rho_0 = 1 - \rho$
5. Probability of n jobs in the system: $\rho_n = (1-\rho)\rho^n$, $n = 0, 1, \ldots, \infty$
6. Mean number of jobs in the system: $E[n] = \rho/(1-\rho)$
7. Variance of number of jobs in the system: $\text{Var}[n] = \rho/(1-\rho)^2$
8. Probability of k jobs in the queue:
$$P(n_q = k) = \begin{cases} 1-\rho^2, k = 0 \\ (1-\rho)\rho^{k+1}, k > 0 \end{cases}$$
9. Mean number of jobs in the queue: $E[n_q] = \rho^2/(1-\rho)$
10. Variance of number of jobs in the queue:
$$\text{Var}[n_q] = \rho^2(1+\rho-\rho^2)/(1-\rho)^2$$
11. Cumulative distribution function of the response time:
$$F(r) = 1\text{-}e^{-r\mu(1-\rho)}$$
12. Mean response time: $E[r] = (1/\mu)/(1-\rho)$
13. Variance of the response time: $\text{Var}[r] = \dfrac{1/\mu^2}{(1-\rho)^2}$
14. q-Percentile of the response time: $E[r]\ln[100/(100-q)]$
15. 90-Percentile of the response time: $2.3E[r]$
16. Cumulative distribution function of waiting time:
$$F(w) = 1 - \rho e^{-\mu w(1-\rho)}$$
17. Mean waiting time: $E[w] = \rho \dfrac{1/\mu}{1-\rho}$
18. Variance of the waiting time: $\text{Var}[w] = (2\text{-}\rho)\rho/[\mu^2(1-\rho)^2]$
19. q-Percentile of the waiting time:
$$\max\left(0, \frac{E[w]}{\rho}\ln[100\rho/(100-q)]\right)$$
20. 90-Percentile of the waiting time: $\max\left(0, \dfrac{E[w]}{\rho}\ln[10\rho]\right)$
21. Probability of finding *n or more* jobs in the system: ρ^n

Table 1: M/M/1 Queue, adapted from [JAIN91].

G. SUMMARY

This chapter has explained simulation in general through the use of a military communication and information system example. This example was then expanded to outline discrete event simulation. Next, the Monte Carlo method of experimentation was introduced. A brief explanation of random variates was then provided, again using our example. Finally, the last section introduced the concepts of queueing theory. The next chapter will discuss the simulation model used to conduct the research for this thesis. The structure of this model will highlight the important concepts summarized above.

IV. THE SIMULATION MODEL

A. INTRODUCTION

This chapter describes the research problem this thesis attempts to answer and explains how the simulation model was built. Section B discusses the model in general and lists the assumptions that were made. Detailed descriptions of the problems outlined in Chapter I are provided in Section C. Section D provides insight into the Silk modeling software and why the JAVA programming language is used. Section E contains a summary and concluding remarks.

B. BACKGROUND

As outlined in Chapter I, we are researching two important problems that, when solved, can be used to enhance the performance of the scheduling mechanism of a resource management system (RMS). The first problem is to find the optimal point at which to schedule a group of jobs waiting to enter the heterogeneous computing system. The second problem is to then ascertain whether there is a point beyond which the relative benefit of group scheduling substantially decreases as the group size increases. The kinds of systems for which these problems are important are both varied and complex. For this reason, this thesis makes various simplifying assumptions to reduce the model's complexity while preserving its substance.

The system being studied can be represented as a sequence of queues and servers used to schedule jobs over an array of heterogeneous computational resources. (See Figure 6.) The system begins by collecting jobs as they arrive until both a predefined group of jobs has been collected and the scheduling server is free to compute the next schedule. The group of jobs is then dequeued and processed by the scheduling server. Upon completion of the scheduler, the jobs are released as a group to their assigned machines for processing. However, prior to being serviced by their assigned machine,

these jobs may be held in an admission queue if the assigned machine is too busy[6]. Thus, the overall system is a single-queue, single-server component feeding into an array of additional single-queue, single-server components.

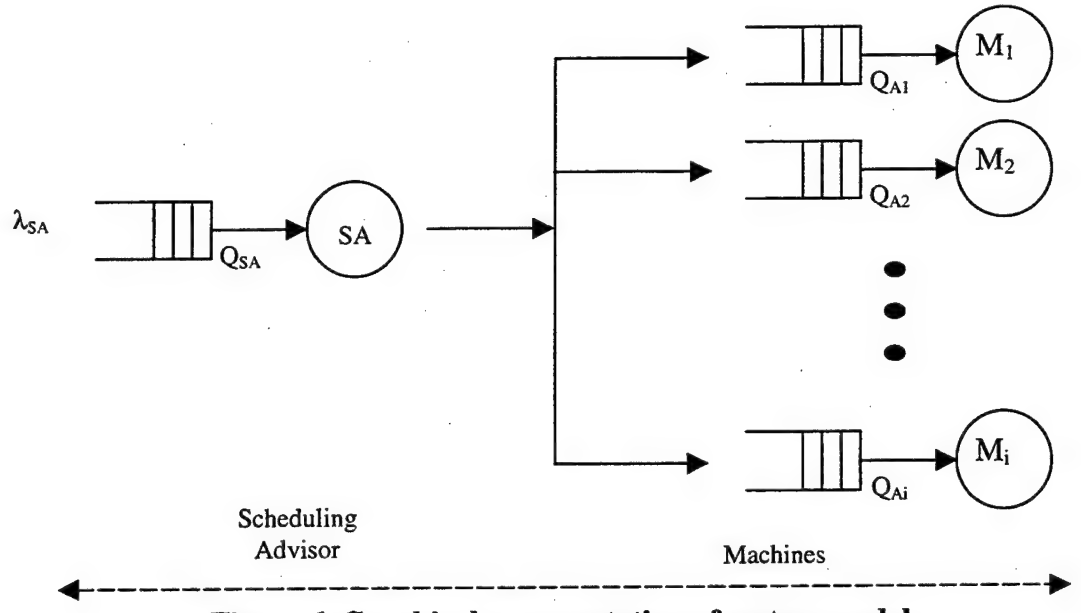


Figure 6. Graphical representation of system model.

In this model there are a total of M+1 queueing systems divided into two types. The first type is located on the left side of the figure and represents the Scheduling Advisor (SA) as described in Chapter II. The remaining M queueing systems belong to the various machines upon which the jobs will be executed. These are called the machine queueing systems and are referred to as M_i where i is the unique index of the machine. Each machine queueing system consists of an admission queue (Q_{Ai}) that holds jobs until the machine is free, and the server, which is the machine itself. [KIDD99]

To begin, we need to define what is a schedule. In this research, a schedule is an assignment of jobs to available machines such that a job runs on exactly one machine and a machine can run no more than one job at the same time. In other words, in this simplified model[7] the machines are not multitasking. The type of scheduling done by

[6] Typically MSHN will not use these queues unless the machine is so overburdened that the OS will reject any new process requests. However, modeling the ready and waiting queues maintained by the OS are beyond the scope of this thesis (see Chapter VII).

[7] Again, this model differs substantially from the environment where MSHN will run. However, this model must be clearly understood and validated before more sophisticated models are analyzed.

this system is non-preemptive. With non-preemptive scheduling, once a job has begun execution, it cannot be interrupted. Lastly, rescheduling is not considered. Rescheduling means that if a job has been assigned to a machine but has not begun execution (i.e., it is waiting in the admission queue), it could be removed from the admission queue, placed back in the SA queue, and rescheduled at the next scheduling event. For this research, once a job has been assigned, it remains in the admission queue until it is executed on its assigned machine.

The QoS metric used in assessing the goodness of a schedule in this model is the average time in the system per job. This metric was chosen because it is readily available, easily manipulated, and can be easily expanded to incorporate other metrics such as priority or security.

The schedules are built using an Expected Time to Compute (ETC) matrix. ETC(j,m) is the expected time for job j to complete execution on machine m assuming that there are no other jobs using any of the machines. In the ETC matrix, the numbers across a row are the execution times of the corresponding job on the different machines. Based upon the variations between corresponding rows and columns, the ETC matrix can be classified into one of four heterogeneity classes. The variation across a row of the matrix is referred to as the machine heterogeneity, while the variation down a column is the task heterogeneity [ARMS97]. Based upon a classification of high versus low heterogeneity, the four classes proposed by [ARMS97] for ETC categories are: (1) high task and high machine heterogeneity (HiHi), (2) high task and low machine heterogeneity (HiLo), (3) low task and high machine heterogeneity (LoHi), and (4) low task and low machine heterogeneity (LoLo). The ETC matrix can be further classified into two additional classes, consistent and inconsistent. For a consistent ETC matrix, if one job has a lower execution time on machine m_x than on machine m_y, then the same is true for all jobs [ARMS97]. Consequently, any ETC matrix that is not consistent is inconsistent. For this model, inconsistent ETC matrices are used because they are likely to arise in a typical environment [MAHE99]. A HiLo ETC matrix is used in the model because it most closely represents the realistic environment depicted in the military scenario outlined in Chapter V.

For measurement and comparison between differing schedules, a schedule's expected completion time is defined as the time at which the last job is expected to complete when all the jobs are executed according to the defined schedule (that was based upon the values in the ETC matrix). Thus, an optimal schedule is defined as the one with the smallest completion time. Lastly, the schedules are constructed assuming that the jobs are independent of each other. In other words, each job executes without needing information from any other job or from the outside environment.

A scheduling algorithm is needed to create a job schedule using an ETC matrix. The problem of finding an optimal schedule where the finishing time is as small as possible for large data sets is known to be NP-complete[8] [IBAR77]. Thus, a common choice is to instead use a heuristic greedy algorithm. Greedy algorithms make locally optimal choices in the hope that these choices will result in a close to optimal solution. The specific algorithm used in this model is a MaxMin algorithm that has complexity $O(MJ^2)$ where M is the number of machines available and J is the number of jobs to be scheduled. This algorithm does not guarantee an optimal solution, but often gives a near-optimal solution. Figure 7 gives simple pseudocode for the MaxMin algorithm.

[8] NP-complete means that it is unlikely that a polynomial time-bounded algorithm exists for this problem.

```
procedure MaxMin;

Input: The ETC matrix, list of jobs submitted to be scheduled.
Output: The schedule that gives an assignment of jobs to machines that attempts to minimize the total
        completion time.

begin

1. Initialize.
        Matrix soonestDone ← matrix ETC;
        Row Vector accumulatedtimes ← 0;
        Mark all jobs as unchosen;

2. While there is still a job that is unchosen do
    begin
        Find the minimum of all rows in soonestDone;
        Find the maximum minimum of the values found above. Let the row index of
                The minimum be jobChosen and column index be machineChosen.
        Assign jobChosen to machineChosen.
        Store assignment (jobChosen, machineChosen);
        Update accumulatedTimes:
                accumulatedTimes[machineChosen] += ETC[jobChosen] [machineChosen];
        Remove the row jobChosen from soonestDone and ETC.
        Update soonestDone:
                Set each row of soonestDone ← Corresponding row of ETC + accumulatedTimes, added entry
                wise.
        Mark the job jobChosen as chosen.
    end.

3. Output all of the assignments (job, machine).

end.
```

Figure 7. Pseudocode for MaxMin algorithm, from [JANA96].

In MaxMin, the ETC matrix is scanned and the minimum value from each row is recorded. Next, the maximum value from the recorded minimums is chosen and that (job, machine) pair is recorded as a schedule assignment. Then, the values in the chosen machine column of the ETC matrix are updated to indicate the new completion time values and the remaining jobs (rows) are scanned and scheduled accordingly. MaxMin iterates as many times as there are jobs, and in each iteration a (job, machine) pair is chosen so that the selected pair contributes to reducing the time required to execute the resultant schedule. Finally, all selected pairs are output as the assignment schedule.

The scheduling system of Figure 6 is simulated using a discrete event simulator such as that described in Chapter III. The job arrivals are modeled using a Poisson random process with the mean interarrival time set by the user. The simulator contains

an ETC matrix generator that creates a HiLo, inconsistent matrix as described previously using both the size of a group of jobs and the number of machines input by the user.[9] Using a MaxMin heuristic, a schedule is then created from the ETC matrix, the real time to compute the schedule is recorded and the jobs are assigned to their respective machines in accordance with the generated schedule. The amount of real time used to create the schedule is then used to advance the simulation time for the scheduler. The jobs may then wait in a FIFO admission queue for their machine until that machine is available to process them. Because the actual execution time of a job can be different from the value given by the ETC matrix, the simulated execution time for each job is modeled by sampling a Gamma probability density function with mean equal to the ETC's expected execution time of the job. The amount of simulated time that each job spends in the system is recorded for later analysis.

C. DETAILED PROBLEM DEFINITION

From the terminology established in Chapter III and the problem description of the previous section, we can now precisely describe the system model. From previous descriptions, the SA queueing system can be classified as a type $M/D^{[G]}/1$ queueing system because the arrivals are exponentially distributed and the time spent in the scheduler is deterministic and based upon the number of jobs in a group and the number of machines in the system. The superscript "[*G*]" indicates that the SA is servicing a group of jobs. Each system of the array of machine queueing systems on the right side of Figure 6 can be similarly classified as $G/E_k/1$ because the arrival distribution at each admission queue cannot be definitely determined and the service time is modeled using a Gamma distribution.

From the above classifications, we can now make specific assumptions about the behavior of these systems in order to obtain a theoretical approximation for the systems' behavior. A significant simplifying assumption is that the runtimes of each job on any of the machines is identical. From this assumption, we can then aggregate the machine queueing systems into a single aggregate system for analysis. Of particular note is the

[9] This procedure is equivalent to selecting a group from a very large population of jobs.

fact that this one aggregate queueing system does not actually reproduce the behavior of the original parallel system, but instead provides a limiting "best" case. This simplifying assumption results in the system shown in Figure 8.

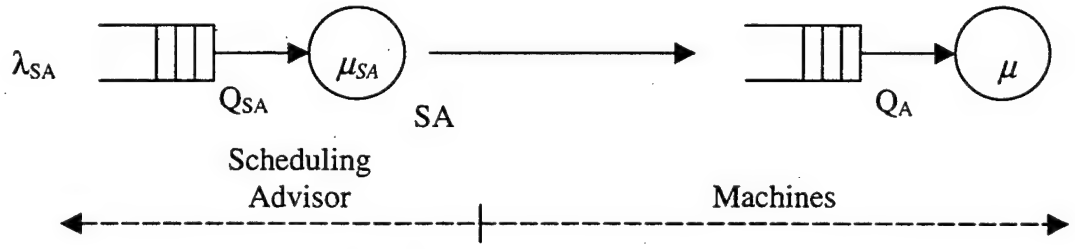


Figure 8. Aggregate queueing model, from [KIDD99].

From this assumption, we can begin to formulate the equations for an approximation to the expected average amount of time spent by a job in the simulated system. From the beginning, since a group of jobs is collected in Q_{SA} before being submitted to the SA, the wait time spent collecting a group of size G is computed by equation (1).

$$w_{Q_{SA}} = \frac{G}{2\lambda_{SA}} = \frac{G\tau_{SA}}{2} \tag{1}$$

Once the group of jobs is submitted to the SA, the time spent by the SA to determine a schedule is simply s_{SA} or $1/\mu_{SA}$. If we define the average execution time for a job on a machine as s, then the average rate of jobs being executed by a machine is expressed as $1/s$ or μ. Hence, the total service rate for M machines is $M\mu$. Since an entire group of jobs is submitted from the SA to Q_A, the average amount of time a job spends in Q_A can be computed from equation (2).

$$w_{Q_A} = \frac{G}{2M\mu} = \frac{Gs}{2M} \tag{2}$$

Similarly, the time required for M to execute a single job on average is simply $1/M\mu$ or s/M. If we add all of the previous intervals together, we can use equation (3) to compute an initial estimate for system response time, or the average total time spent by a job in the simulated system.

$$r_{system} = \frac{G}{2\lambda_{SA}} + \frac{1}{\mu_{SA}} + \frac{G}{2M\mu} + \frac{1}{M\mu} = \frac{G\tau_{SA}}{2} + s_{SA} + \frac{Gs}{2M} + \frac{s}{M} \tag{3}$$

It is important to note here the simplifications inherent in this estimate. Equation (3) does not consider the complexities of the actual expressions for s_{SA} or s/M. For example, considering the mean service time for the SA, which is dependent upon G, M, and the algorithm used to schedule, we can see that $1/\mu_{SA}$ will be $O(MG^2)$. This assumption is based upon the fact that the scheduler uses a Greedy algorithm with complexity $O(MJ^2)$. Therefore, equation (3) can be rewritten as equation (4).

$$\frac{G}{2\lambda_{SA}} + \left(a_4 G^2 M + a_3 GM + a_2 G + a_1 M + a_0\right) + \frac{G}{2M\mu} + \frac{1}{M\mu} \tag{4}$$

Using equation (4) as a starting point and including the results from some experimentation with the simulation, the a_i coefficients can be determined. The complexities introduced as a result of grouping jobs may demonstrate that the above equation is too simplistic and that the assumptions made reduce the granularity of the problem to a level where the above equation is actually a poor estimate. However, through the development of this equation, we have begun the analysis as to how changes in G will affect the average time that a job spends in the system. Additionally, we can learn from these equations how large G must become before the relative benefit of improvements in the schedule will be overcome by the time spent waiting for the jobs in the group to collect. [KIDD99]

D. JAVA AND SILK

Once the form of the model was decided, and we had determined that, because of the model's complexity, simulation was needed, the next step was to choose a programming language for the simulation. Primarily we desired an object-oriented, powerful language with multithreading and an easy reuse capability. When Sun Microsystems introduced JAVA in 1995, they described it as "a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language" [FLAN97]. Because JAVA is an object-oriented programming language with robust reuse capabilities, it was an easy choice as the language for this project. Due to a desire for ease of development, finding a simulation tool based on the JAVA language was the next step. Our motive for finding a discrete

event simulation tool was to be able to work within an environment where event scheduling, thread handling and random variate generation were provided and therefore wouldn't have to be developed. A search of the Internet revealed the Silk simulation tool.

Silk is simply a collection of JAVA classes for discrete event simulation. These classes include an entity simulation engine using the multithreaded power of JAVA and a set of process-oriented modeling methods for object-oriented simulation design. The combination of Silk and JAVA provide an integrated simulation and programming environment for building reusable modeling components. Additionally, JAVA's inherent support for multithreaded execution and Silk's incorporated event scheduler produce the essential ingredient for representing the concurrent flow of entities in process simulation models. Lastly, because Silk is simply an extension of JAVA, the full flexibility of the JAVA programming language is available allowing the user to create more complex models by creating customized extensions to the Silk classes. Additional information about the Silk simulation tool can be found at www.threadtec.com. [KILG98]

E. SUMMARY

This chapter has described this research's simulation model. It explained the simplifying assumptions used to build the simulation model and described the simulation steps. Next, the research problem was described in detail. Finally, the last section explained why JAVA was chosen as the programming language for the model and why the Silk modeling software was also chosen. The next chapter explains how experiments were conducted using the simulation model described in this chapter.

V. EXPERIMENTS

A. INTRODUCTION

This chapter details the experiments conducted using the simulation model. Section B provides a military scenario where a system requiring answers to the questions posed may be employed. The experiments were performed using this scenario. The data resulting from the experiments are presented in Section C. Section D outlines the validation of the simulation model and the experiment. A summary is provided in Section E.

B. BACKGROUND

In an effort to provide realism both to the experiment and for the use of a Resource Management System (RMS) in a heterogeneous environment, we developed a military scenario embodying the parameters expected in a practical application. The following parameters are needed for this simulation experiment:

- M - the number of machines in the system available for processing jobs,
- τ - the mean time between successive job arrivals (in milliseconds),
- s - the mean duration expected for a job to execute on a machine (in milliseconds), and
- G - the size of a group of jobs to be scheduled.

With these parameters in mind, the scenario is based on a US Marine infantry battalion (Bn) that is ready to begin the combat portion of an exercise. The Bn headquarters (HQ) has a networked system of 16 computers, one of which is the resource scheduler. This system of computers consists of Wintel 486 machines, older Wintel Pentium machines and Sun SPARC machines. The overall processing power of all these machines is considered to be approximately the same, and the types of jobs submitted to the system are very diverse[10]. Thus, the resultant networked system consists of 15 heterogeneous computers available to process many different types of jobs. The majority of jobs

[10] For these reasons, a HiLo Expected Time to Compute (ETC) matrix, as defined in Chapter IV, is used to represent the computing environment for this scenario.

processed within the Bn are not complex and it is assumed that on the average a job can be completed within one second (1000 ms), regardless of the type of processor used. It is also assumed that just prior to entering a combat exercise, the level of computational activity for the Bn rises dramatically and that the mean interarrival time for jobs to the RMS is between 50 and 100 ms during this time.[11]

Before outlining the methods for the experiment, we must first describe the platform used to conduct the simulation. The simulation experiments were all executed on a Silicon Graphics (SGI) Challenge-L multiprocessor machine named *caesar*. *Caesar* has four 200MHz MIPS R4400 processors using the IRIX64 operating system, version 6.2, with JAVA version 3.1.1 (Sun 1.1.6). *Caesar* was accessed remotely from an Intel Pentium II, 400MHz, single processor machine using the Windows NT Workstation 4.0 operating system.[12]

Using the parameters from the above scenario, the first major decision was the choosing of a job interarrival time that would provide experimentally significant results. Although this system involves both multiple queues and servers, we borrowed a stability condition relationship from single queue, single server systems. A system is said to be unstable if the average number of jobs in the system during any reasonable interval grows continuously. Thus, for stability, the mean arrival rate should be less than the mean service rate (i.e., $\lambda < M\mu$) or equivalently the traffic intensity, $\lambda/M\mu$, must be less than 1 [JAIN91]. Therefore, although our system is more complex than a single queue, single server system, we chose an interarrival time of 67ms satisfying this general stability condition.[13]

---

[11] This range of interarrival times was chosen to satisfy the stability condition for the system. The significance of this range will be outlined later and in Chapter VI.

[12] The Intel Pentium II machine was not used to conduct the experiment because the required millisecond time resolution was not available. The Java Virtual Machine (JVM) and Windows NT have a compatibility problem that only permits 10ms resolution. Similarly, there is a thread "suspend/resume" compatibility problem between the JVM and Sun Solaris that causes a multithreaded JAVA program to "hang." Therefore, the experiments were conducted on the SGI platform because millisecond resolution was provided and the program did not "hang."

[13] For this experiment, M=15, μ=0.001 and λ=0.0149. Thus, $\lambda/M\mu = 0.0149/0.015 = 0.993 < 1$.

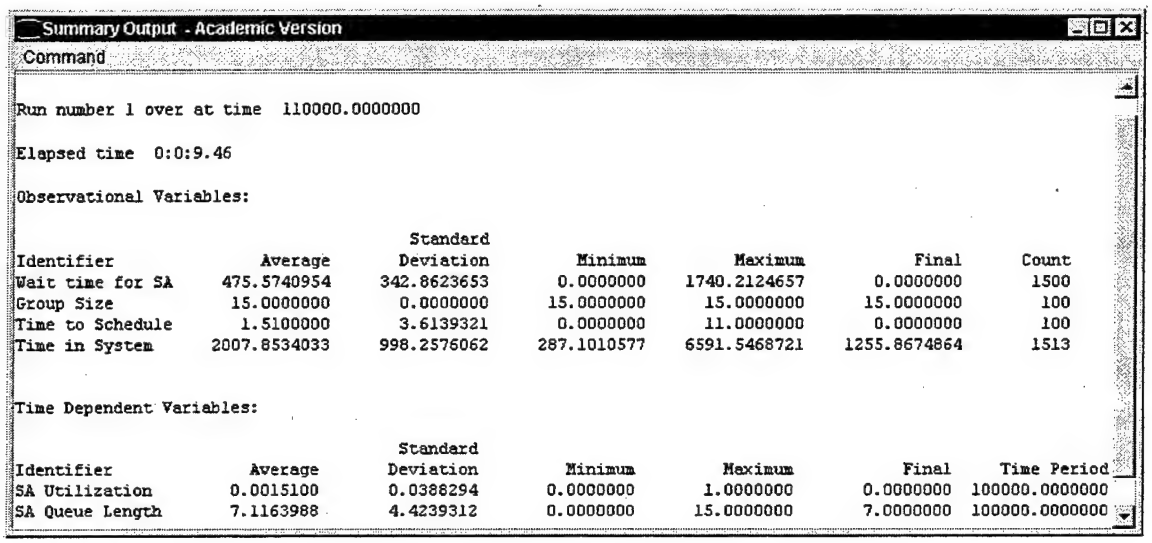
With this initial parameter set, we next established the methodology. The general experiment was controlled using time. Over an interval of 110 simulated seconds, the amount of time each job spent in the system was recorded. These results were normalized giving the average time spent in the system per job. In order to capture results from a system already in operation (i.e., in a steady state), statistics were not recorded until after 10 seconds of simulation time had elapsed. The initial 10 seconds permitted the simulation to reach a steady operating condition so that the skewed data from the simulation's "start-up" was avoided. Experimentation and analysis verified the steady state condition. Because we were interested in how the size of the group of jobs, *G*, affected the Quality of Service (QoS) measure, namely the average time in the system per job for this research, we ran the experiment 30 times for each choice of *G*. The results of these 30 runs were then averaged to find the overall mean time spent in the system per job for each value of *G*.

An additional experiment included running the simulation with different inputs for both the number of jobs and machines, and then recording the amount of time taken for the scheduler to compute a schedule. The data from these experiments were used to calculate the coefficients for equation (4) in Chapter IV. The next section provides the results of the experiments conducted using the scenario introduced at the beginning of this chapter.

C. RESULTS

To briefly review what has been discussed thus far, this research studies the affect of grouping jobs prior to scheduling upon the average time that a job spends in a heterogeneous computing environment managed by an RMS. The experiment is conducted using a computer simulation model of the system. The simulation is coded in the JAVA programming language using the Silk simulation building tool. The parameters for the experiment are established using a military scenario involving an infantry battalion just prior to a combat exercise. For each value of *G* (the size of a group of jobs) the simulation is run 30 times and the mean interval that a job spends in the system is calculated. Figure 9 illustrates the output received from each run of the experiment.

The results of the experiment using the scenario's parameters (M = 15, τ = 67ms, s = 1000ms) are provided in Table 2 and graphically shown in Figure 10. From these results, it is apparent that there is a positive and definite benefit gained when jobs are scheduled in groups (from the additional information available to the scheduler). In fact, for this scenario and set of parameters, the optimal point to initiate a scheduling event is when the first of either of two events occurs: when the SA queue reaches a length of 17 or after 1139 milliseconds (67ms X 17 jobs) have transpired.



Summary Output - Academic Version

Command

```
Run number 1 over at time  110000.0000000

Elapsed time  0:0:9.46

Observational Variables:

                                  Standard
Identifier              Average   Deviation      Minimum       Maximum         Final       Count
Wait time for SA     475.5740954  342.8623653    0.0000000  1740.2124657    0.0000000        1500
Group Size            15.0000000    0.0000000   15.0000000    15.0000000   15.0000000         100
Time to Schedule       1.5100000    3.6139321    0.0000000    11.0000000    0.0000000         100
Time in System      2007.8534033  998.2576062  287.1010577  6591.5468721 1255.8674864        1513


Time Dependent Variables:

                                  Standard
Identifier              Average   Deviation      Minimum       Maximum         Final     Time Period
SA Utilization        0.0015100    0.0388294    0.0000000    1.0000000    0.0000000  100000.0000000
SA Queue Length       7.1163988    4.4239312    0.0000000   15.0000000    7.0000000  100000.0000000
```

Figure 9. Summary output for simulation experiment.

Group Size	Avg Time/Job
1	3196
3	2899
5	2623
7	2502
9	2421
11	2178
13	2104
15	1885
17	1872
19	1971
21	2065
23	2112
25	2243

Group Size	Avg Time/Job
27	2368
29	2534
31	2682
33	2828
35	2954
37	3108
39	3216
41	3346
43	3500
45	3640
47	3773
49	4061
51	4152

Table 2: Results for M=15, τ=67ms.

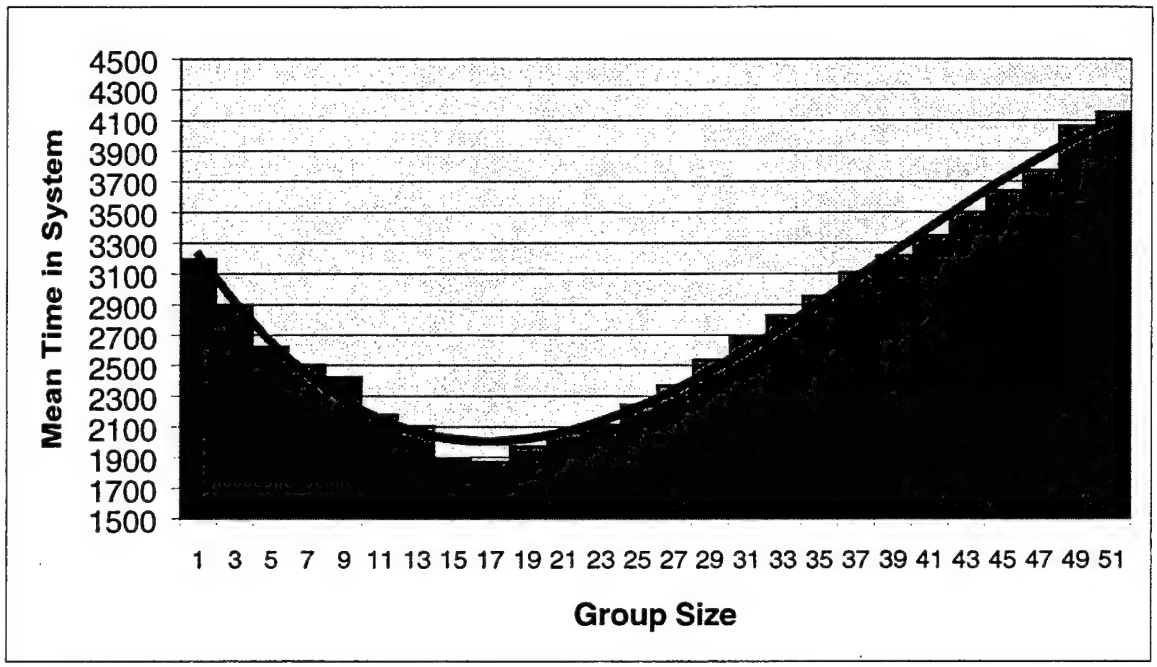


Figure 10. Graphical results for M=15, τ=67ms.

The data in Figure 10 also demonstrates the significance of the 67ms interarrival rate. The data infers that with this mean job interarrival rate the system can optimally handle a "group" interarrival rate of 1139ms (67ms X 17 jobs) to the scheduler. The intuitive implication of this data is that group sizes below 17 cause either an inefficient schedule to be created or a system bottleneck within the array of processing machines admission queues. Additionally, group sizes above 17 do not provide a scheduling benefit greater than the amount of time taken to gather a group. Further analysis of these results requires additional experimentation and data collection. This topic is explored further in Chapter VI.

G	M	s_{SA}(Time to Schedule)
100	100	275
50	100	65
10	100	3
100	10	79
50	10	21

Table 3: Results from scheduling experiment.

The results from the second experiment are provided in Table 3. In this experiment, the simulation was run for the values of G and M shown in the table; the mean time to compute a schedule was recorded. Recalling the term for the response time of the SA in equation (4) of Chapter IV, the formula for just the scheduler in terms of G and M can be written as equation (5).

$$s_{SA} = a_4 G^2 M + a_3 GM + a_2 G + a_1 M + a_0 \tag{5}$$

Using the data from Table 3 and substituting into equation (5), the solution to the resultant system of five equations with five unknowns reveals the coefficients for equation (5) in Table 4. Because the platform where the experiment was performed (*caesar)* did not provide enough time resolution, equation (5) was evaluated for larger values of G and M, resulting in less accurate s_{SA} values for small G and M. Even so, at those smaller values s_{SA} is two orders of magnitude smaller than the overall result and as such was not significant to this research. Of particular note is the fact that this solution is only valid for the platform on which the experiment was run and the scheduling algorithm that was used. Thus, this particular solution to calculate the estimated response time of the scheduler is only valid for simulation experiments conducted using a greedy scheduling algorithm executed on the SGI machine named *caesar* mentioned previously[14].

a_0	a_1	a_2	a_3	a_4
-25	49/180	37/45	-187/18000	53/180000

Table 4: Coefficients of equation (5).

With the results from the simulation experiment presented, the next section compares the experimental results to the theoretical, expected results.

[14] This analysis was done because the simulated time for the scheduler was advanced according to real time as outlined in Section B of Chapter IV.

D. VALIDATION

In order to validate the simulation model, we used two primary techniques. The first technique involved using hand calculations to determine expected results and comparing the experimental results against those predictions. This technique was used to validate the ETC matrix generator and the MaxMin Greedy scheduler. The second method involved comparing the results from the simulation against expected theoretical results. This technique was used to validate the results of the simulation experiment as a whole with the aim of determining whether the mean response time per job from the simulation were comparable to theoretical expectations.

First, we verified the ETC matrix generation routine. Using a mean job execution time of 100ms, a job heterogeneity factor of 0.5, and a machine heterogeneity factor of 0.1, the matrix in Table 5 was generated. Inspection of this matrix confirms the operation of the generation routine in this one example. The heterogeneity down the rows is high, while the heterogeneity across the columns low. Thus, a HiLo, inconsistent matrix was created.

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
JOB #0	133	136	157	179	130	139	120	136	131	162
JOB #1	176	219	178	221	222	181	197	192	175	218
JOB #2	29	37	35	35	25	35	34	29	26	32
JOB #3	21	19	19	27	20	21	22	21	21	20
JOB #4	35	32	36	30	34	29	34	31	35	40
JOB #5	124	134	129	114	116	132	114	117	128	120
JOB #6	94	103	88	108	104	99	96	115	92	99
JOB #7	178	194	210	172	182	181	159	163	169	186
JOB #8	135	126	129	126	138	119	132	141	112	107
JOB #9	71	75	71	75	62	76	61	78	77	83

Table 5: Sample HiLo ETC matrix.

Once the matrix generator was verified, we validated the MaxMin scheduling routine. This task was done by generating a series of matrices and sending them to the scheduler. The results from the scheduler were recorded and corroborated against hand calculated schedules created using the algorithm presented in Figure 7. The two sets of schedules matched, confirming the MaxMin scheduling procedure.

To validate the overall operation of the simulation, we must apply reasoning and determine whether the results achieved from the experiment are similar to theoretically expected results. Two very significant simplifying assumptions were made about our system in Section C of Chapter IV. The first was that the response time of the entire system was simply a summation of the response time of the SA portion on the left of Figure 6 and the machine portion of the system on the right. The second was that the response time of the machine portion of the system could be simply represented by $1/M\mu$ or s/M.

The first simplification is actually only valid for a system where jobs are scheduled individually as they arrive. In this case, a job would arrive, be serviced by the scheduling server when available, be forwarded to the admission queue of its assigned machine and ultimately be processed in FIFO order. For this situation, the summation of the various queue wait times and service times of the servers would be valid. However, the act of grouping the jobs and submitting them collectively to the scheduler, but then assigning them individually to machines introduces complexities to the system that preclude a simple summation of individual response times. The state of the system relative to the size of the group of jobs becomes a significant factor. Therefore, a more appropriate method of analysis is to consider the SA portion and the machine portion of the system separately and determining the relationship between them when jobs are grouped.

Before considering the relationship between the two sections of the system, the actual form of the equation for the response time of the machine portion of the system must be found. In [IBAR77], Ibarra and Kim derive a lower bound for the service rate of a set of heterogeneous machines, namely the sum of the minimum runtimes of each job taken across all machines divided by the number of machines, M. Therefore, in a simplified form, s/M can be considered the lower bound for the service time per job for the array of machines. Additionally, we assume that sophisticated RMSs use schedulers that perform better, or at least as well, when more information is available (i.e., a larger G is used). Therefore, the service time across the array of machines is also dependent upon the scheduling algorithm and G, the size of the group scheduled. Typically, as G becomes larger, more information is available, and a better schedule can be produced

[KIDD99]. Combining all of these derivations and assumptions, we can theorize that the service rate per job for the array of machines is greater when G is 1 and approaches a lower bound asymptotically as G gets larger. Thus, as depicted in Figure 11, the expected form of this term is exponential in nature and decreases asymptotically to the lower bound as G gets larger.

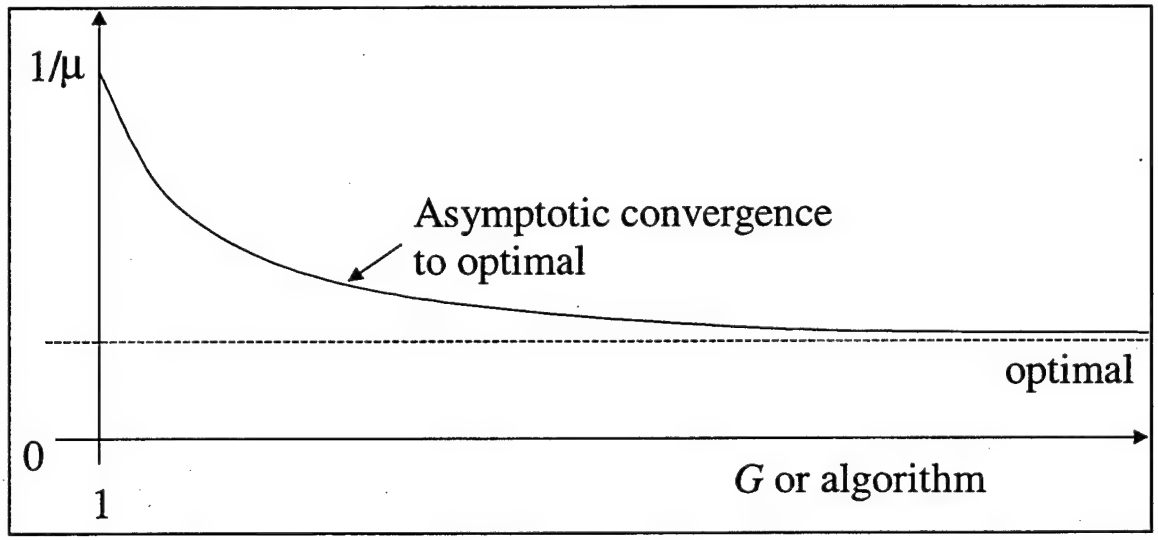


Figure 11. Average runtime across machines vs. larger *G* or better algorithm.

Since the form of the simplified equation for the service time of the machine array has been determined, we now infer how job grouping increases the complexity of the relationship between the SA and the machine parts of the system. Because the results from our experiment show that G should remain relatively small, we find the effect of the service time for the scheduler is essentially zero in comparison to the other terms, and so, can be ignored. Therefore, the response time for the SA system is approximately equal to the wait time in the SA queue and while G is small, this queueing time is insignificant. (It is generally two orders of magnitude smaller than the total time spent in the system.) During this situation, the dominant factor of the system must then be the service time across the machine queue system on the right of Figure 6. As G increases, the mean response time for the entire system will continue to decrease nearly exponentially (to the localized minimum) until the point when the wait time in the SA queue surpasses the relative benefit achieved from having a better schedule. At this point, the dominant factor in the system becomes the wait time in the SA queue instead of the service time of the latter machine part of the overall system. Once this point is reached, the mean

response time of the overall system is expected to increase linearly from the localized minimum as the size of *G* increases (and hence the wait time in the SA queue increases). (See Figure 12). In summary, the response time relationship between the SA system and the machine queue system is not a simple summation, but is rather exponentially decreasing while *G* increases until a local minimum is reached, at which point the relationship becomes linearly increasing as *G* continues to increase.

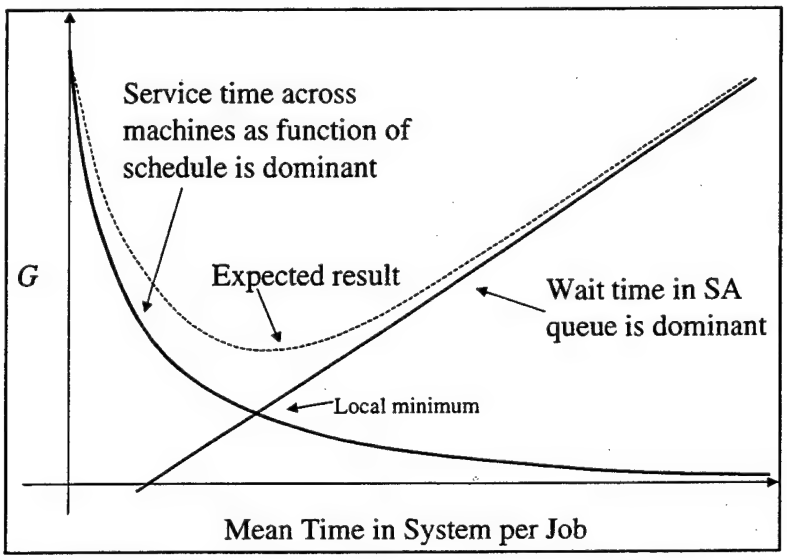


Figure 12. Dominant factor of system as *G* increases.

To test the validity of the model, the theoretical predictions are compared to the experimental results. Using the results from the experiment, a best-fit regression analysis was done for the data where *G* ranged between 1 and 17. The regression plot showed that an exponential regression is the best fit to the data. (See Figure 13.) A second best-fit regression analysis for the data where *G* ranged between 17 and 51 was performed. This regression plot showed that a linear regression is the best fit to the data. (See Figure 14.) The value $G = 17$ was used to divide the two graphs because it was the local minimum in Figure 10. Therefore, because the experimental results correspond to the theoretical expectations detailed previously, we have confidence that the model is accurately simulating the real system.

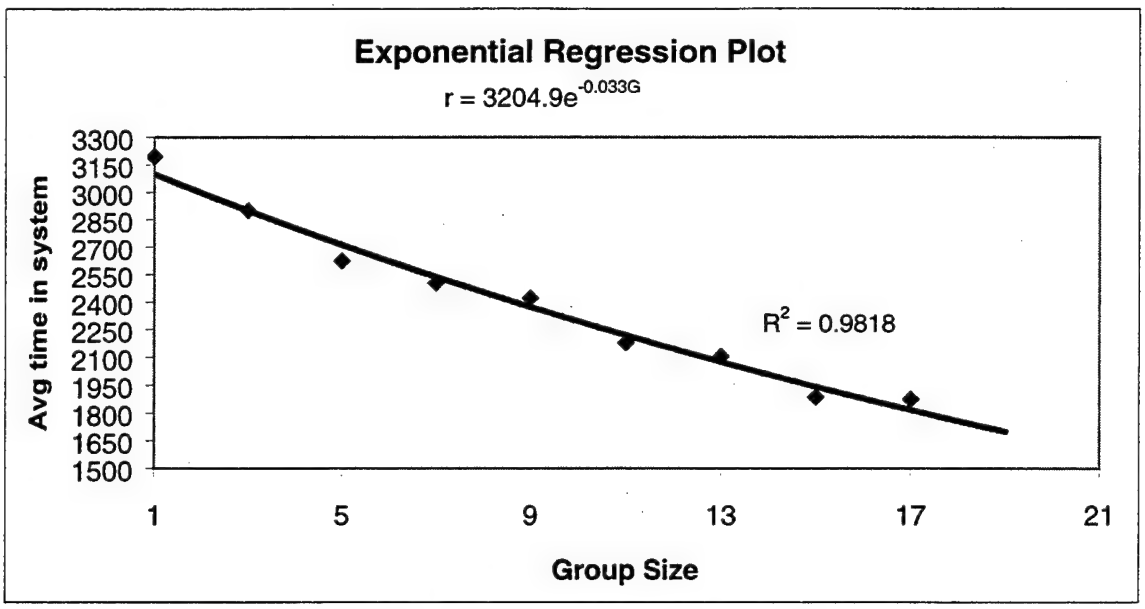


Figure 13. Regression Plot for G < 19.

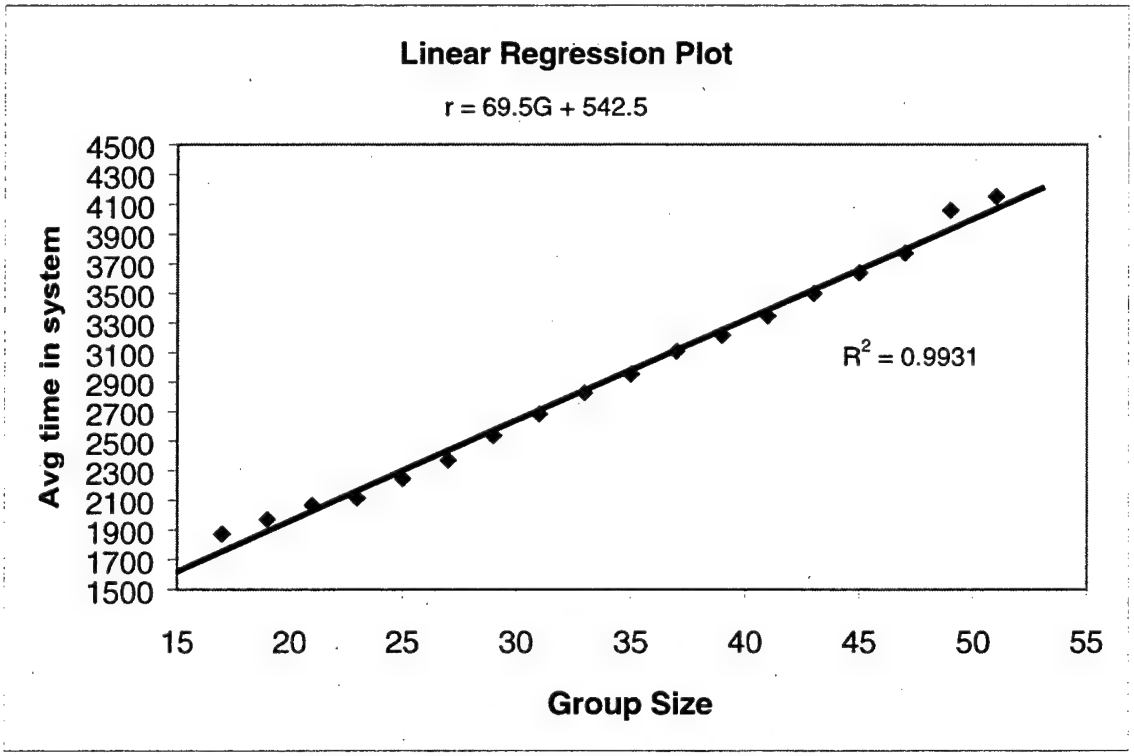


Figure 14. Regression Plot for G > 17.

E. SUMMARY

This chapter described the experiments conducted using the simulation model. It explained the scenarios behind the experiments, which provide the necessary parameters for the investigation. Next, the data resulting from the experiment was presented. The last section explained the validation procedures and a theoretical analysis of the predicted system output. The next chapter will explain the additional analysis required to formulate the cost tradeoff relationship between the potential performance improvement obtained from having a larger pool of jobs to schedule, and the potential performance loss resulting from having to wait for that particular pool size.

VI. ANALYSIS

A. INTRODUCTION

The previous chapters provided background for the research conducted in this thesis. In this chapter, the optimal point at which to initiate a scheduling event for an RMS scheduling mechanism is found. This optimal point can be delineated by the size of a group of collected jobs (G), or by a maximum time threshold if the designated size of the group has not been reached. Using the military scenario described previously, additional experimentation needed to solve the cost tradeoff problem presented in Chapter I is highlighted in Section B. Section C describes the statistical techniques used throughout this research. The results are featured in Section D. The final section will summarize the highlights of this chapter and provide any concluding remarks.

B. BACKGROUND

The experimental results obtained in Chapter V were useful in determining the validity of the model and in showing that changing the group size does affect the average time a job spends in the system. However, additional experimentation is needed to demonstrate how, and under what conditions, the group size influences the performance of the system. From such results, a cost tradeoff relationship can be determined.

Preliminary experiments revealed that the effect of grouping varied, depending upon the job interarrival times (i.e., arrival rate) to the system. This observation led to the use of an important state descriptor for the system called the utilization factor. The utilization factor is defined as the quantity $\rho = \lambda/M\mu$ and is a measure of how heavily the resources of a queueing system are utilized [LAW91]. Continuing to use the same service rate, $M\mu$, from the scenario described in Chapter V, the utilization factor is changed by altering the job arrival rates, λ. In the initial experiment, a utilization factor of 0.995[15] was used as a reasonable state to start the investigation. We found that in this state, the average time that a job spent in the system could be minimized by changing the

[15] The value of 1.0 was desired but from the variables used in the scenario, 0.995 was achieved.

size of the group of jobs collected prior to scheduling. In fact, the optimal group size for our scenario was 17. Using $\rho=1$ as a starting point, the next step was to incrementally reduce the utilization factor by increasing the job interarrival times while repeating the initial experiment to find how changing the group size would influence the average time a job spent in the system. The same procedure was done for an increasing utilization factor. (See Table 6). The data from these experiments is then used to complete the cost tradeoff analysis for the system.

C. STATISTICAL TECHNIQUES

Before presenting the results of the final experiments, this section provides an explanation of the statistical techniques used throughout this thesis. The two primary techniques used are (1) determining a sample mean, and (2) curve fitting.

For a given population, the mean or expected value of the random variable X is denoted by either E(X) or μ and is defined as:

$$\mu = E(X) = \sum_{i=1}^{n} p_i x_i = \int_{-\infty}^{+\infty} x f(x) dx$$

where summation is used for discrete variables and integration used for continuous. However, this research uses the sample mean, which is an estimate for the population mean. The sample mean, $\overline{X}(n)$, is found by taking the sum of all observations, x_i, and dividing this sum by the number of observations, n, in the sample.

$$\overline{X}(n) = \frac{\sum_{i=1}^{n} X_i}{n}$$

Thus, to find the mean time that a job spent in the system, the sum of all the observed times was divided by the number of runs of the experiment (in our case $n = 30$).

The second statistical technique is that of curve fitting. Curve fitting is the process of finding equations of approximating curves that best fit sets of data given a certain metric. For this research, both linear and nonlinear curve fitting are used. The process of curve fitting starts with plotting corresponding values of the variables of interest onto a rectangular coordinate system. This graph is called a scatter diagram.

Often, it is possible to visualize a curve that approximates the data. If the data appear to be well approximated by a straight line, then a good estimate for the data is that the variables have a linear relationship. If the data appear to be better approximated by a curved line, then a curvilinear relationship may be a good estimate. This research includes data that is related in a linear sense (see Figure 13), an exponential sense (see Figure 14), and in a quadratic sense (see Figure 15). The Microsoft Excel and Minitab programs are used for curve fitting in this thesis. A primary reason for curve fitting is regression analysis. Regression analysis allows a random variable to be predicted as a function of another variable. The estimated variable is called the response variable and the variable used to predict the response is the predictor variable or the factor. The overall goodness of a regression is measured by R^2, the coefficient of determination. This coefficient is the fraction of the variation that is explained by the regression. Overall, the higher the value of R^2, the better the regression and "goodness of fit" for the curve fitting the data. For additional information about regression models see [JAIN91].

D. RESULTS

The results from this portion of the thesis generalize the results found in Chapter IV, and used many more experiments involving differing values for the job interarrival time, τ. The point of these experiments was to find the value of G where the average time spent in the system is minimized. For each (τ, G)-pair, the simulation was run 30 times and the mean time spent in the system per job calculated. This step was repeated until a local minimum time is found for the selected interarrival time. The corresponding value of G was then recorded. The results from these experiments are found in Table 2. From this data and the data in Table 6, the equivalent utilization factors are calculated using the parameters from the original military scenario and the different job interarrival rates. The utilization factors are paired with their corresponding group sizes in Table 7. The data from Table 7 is plotted in Figure 15 on a scatter diagram along with the "best-fitting" curve for the data. Using regression analysis, the equation for the relationship between the utilization factor and group size is given by equation (6).

$$G = -104\rho^2 + 208\rho - 86 \tag{6}$$

τ =50, ρ = 1.33	
Group Size	Mean time
3	9067
4	9102
5	8958
6	9041
7	8985
8	8993
10	9077
15	9309

τ = 53, ρ = 1.25	
Group Size	Mean time
10	6835
12	6751
13	6584
14	6303
15	6538
17	6787
20	7310

τ = 89, ρ = 0.75	
Group Size	Mean time
1	1691
5	1650
8	1564
10	1547
11	1533
12	1544
15	1592
20	1911

τ = 100, ρ = 0.67	
Group Size	Mean time
1	1479
3	1493
5	1483
6	1454
7	1447
8	1455
10	1495
15	1628

Table 6: Data for various job interarrival times, τ.

Utilization Factor (ρ)	Group Size (G)
0.67	7
0.75	11
0.995 (1)	17
1.25	14
1.33	5

Table 7: Data for regression plot.

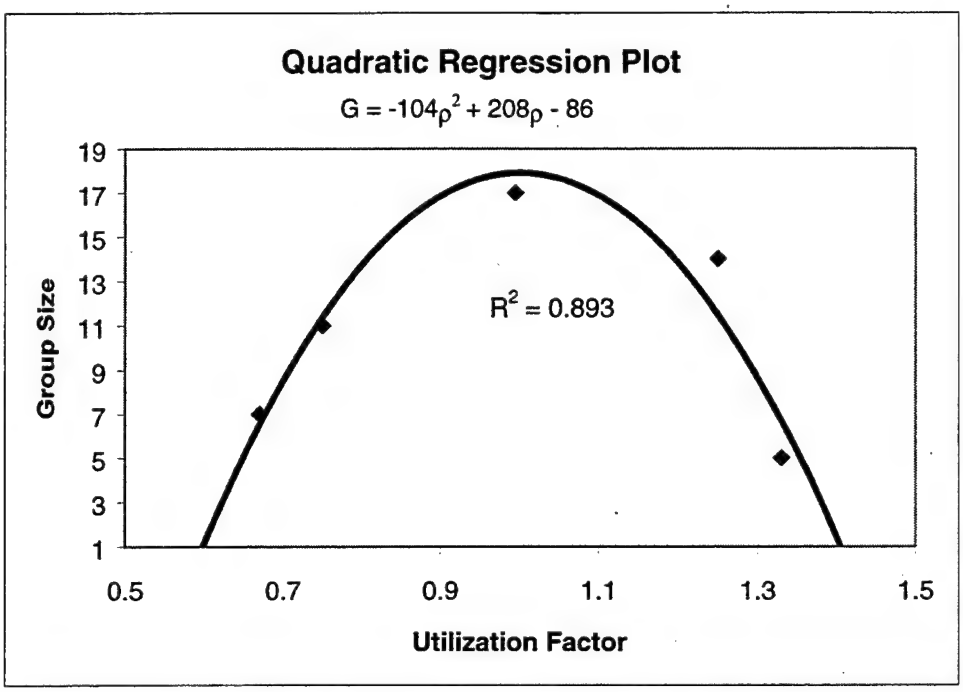


Figure 15. Regression Plot for G as a function of utilization factor, ρ.

Using equation (6) and setting G=1, the utilization range where grouping jobs will minimize the average time a job spends in the system is $0.597 < \rho < 1.403$. Solving equation (6) will provide the group size, G, that results in the minimal time that a single job will spend in the system on average. For the Quality of Service (QoS) measure used in the experiments (i.e., the time a job spends in the system), this result also means that if the utilization factor for the system is outside of the above range, grouping jobs does not provide any better performance then does scheduling jobs individually as they arrive. For example, using the military scenario given in Chapter IV and a known job arrival rate, λ, the utilization factor can be determined and the best group size found.

Thus, the cost tradeoff relationship existing between the potential performance improvement obtained from having a larger grouping of jobs to schedule, and the potential performance loss resulting from waiting for that particular group size to collect is given by equation (6). This equation provides the optimal size of the group of jobs given the current utilization factor for the system. The utilization factor is also the metric to use to determine when the relative benefit of group scheduling substantially decreases as group size increases. For instance, given the scenario provided in this research (HiLo inconsistent ETC matrix with mean service time = 1000ms, MaxMin scheduling

algorithm, $\tau = 67$, and 15 machines), the group of jobs to be scheduled should never be permitted to get any larger than 17 because the amount of time spent collecting a group of that size surpasses the benefit achieved from the improved schedule. Of particular note is the fact that job grouping has the most impact upon minimizing the time a job spends in the system when the utilization factor is near 1.0. As the utilization factor gets further from 1.0, job grouping has progressively less effect until the system reaches a point where grouping is actually detrimental to system performance.

E. SUMMARY

This chapter has presented information about the experiments performed, calculations completed and methods used to determine the relationship between our system's schedule effectiveness and group size. We defined the utilization factor of the system and how this ratio describes the state of the system based upon the job arrival rate, machine service rate, and the number of machines. The statistical techniques used to analyze the data collected from the simulation were also presented. Lastly, we found that the optimal group size to collect for jobs being submitted for scheduling is quadratically dependent upon the system's utilization factor.

VII. CONCLUSIONS AND FUTURE WORK

This thesis accomplished the three main objectives presented in Chapter I. First, Chapter IV described the simulation model built to estimate the solution to our complex analytical problem. This simulation model uses Monte Carlo methods and discrete event simulation. The material provided in the Appendices supplements this description. Second, attempting to minimize the time that a job is in the system as the quality of service (QoS) and a specific scenario, Chapter VI presented a solution to the problems outlined in the first chapter. Finally, this chapter explains how the methodology and results of the experiment can be generalized for any scalar QoS measure. Additionally, this chapter will describe suggested follow-on work.

A. CONCLUSION

Any computer resource management system (RMS), including the Management System for Heterogeneous Networks (MSHN), requires a resource scheduling mechanism. This scheduler must be able to accept arriving resource requests and map them to resources in a manner that attempts to optimize some quality of service (QoS) measure. This thesis presented, in Chapter IV, a simulation model of a resource scheduler that uses a greedy heuristic to minimize the time a job spends in the system. This model was then used to determine the effect that grouping arriving jobs had on the optimization criteria. From experiments using this model, we found a relationship between the state of the system, defined by the utilization factor, and the size of a group of jobs submitted to the scheduler. Using G and ρ to represent the group size and utilization factor, respectively, the relationship we found for a scenario where $\tau = 67$, $M = 15$, $s = 1000$ms, and the ETC matrix is HiLo (inconsistent), is represented by the equation, $G = -104\rho^2 + 208\rho - 86$. From this equation, we also found that job grouping only provides a benefit in terms of our QoS measure when the utilization factor is in the range, $0.597 < \rho < 1.403$. Within this range, the average amount of time that a job spends in the system decreases exponentially as the scheduler group size increases until reaching a minimum dependent upon the utilization factor. Once this local minimum is reached, the time begins to rise linearly (asymptotically) as the group size continues to increase.

When the utilization factor is outside this range, the response time of the system continuously increases as the group size is increased. From the theoretical background presented in Chapter IV, these results indicate that as the group size increases from an initial value of one, the QoS measure is most affected by the throughput across the machines in the system, which is a result of the quality of the schedule. Once the minimal point is reached, increasing the group size simply causes a delay in the queue feeding the scheduler (namely, the amount of time it takes to collect the group of jobs). In turn, this delay simply causes the overall response time of the system to linearly increase. Hence, for any given utilization factor, the respective value of G found from our relationship equation is the point where the wait time required to collect a larger group surpasses the relative benefit achieved from the resulting improved schedule.

Armstrong, in [ARMS97], found that greedy scheduling algorithms performed comparably to more simple algorithms, such as that for Opportunistic Load Balancing (OLB), when in a HiLo (inconsistent) heterogeneity domain and the goal was to finish all jobs in the least amount of time. Thus, the added overhead resulting from using the MaxMin greedy scheduling algorithm is not warranted for the category of heterogeneity in our scenario. However, the positive results we attained when grouping jobs in a HiLo type of scenario indicates that our results will be more significant in a scenario where the performance of the scheduling algorithm is improved. For example, Armstrong found that greedy scheduling algorithms performed significantly better than OLB in a LoHi (inconsistent) heterogeneity domain. The results of this thesis imply that the relative benefit of grouping jobs will be more prominent in a LoHi domain, i.e., when the scheduling algorithm is better matched to the heterogeneity space as outlined in [ARMS97]. The actual results for this matching is a topic for future work.

Although the simulation experiments used an Expected Time to Compute (ETC) matrix and the minimum time a job spends in the system as the QoS optimization criteria, the experiments are not limited to their use. Because all the methods and algorithms in the model use scalar values as decision criteria, the techniques demonstrated in this thesis can be generalized to use any scalar QoS metric. The object-oriented nature of the simulation model allows for easy extension of the base JAVA classes and the substitution of different QoS measures and optimization criteria in the scheduling heuristics. The

backbone of the simulation only requires a matrix of scalar values to optimize in order to create a mapping of jobs to machines. Thus, the values in the matrix could be from any scalar valued QoS metric, and the scheduler could be changed to use any algorithm operating on the matrix. For this reason, only minor changes are needed to use the simulation model built for this thesis to test the effect of grouping resource requests on any RMS using a similar structure and on any defined, scalar QoS metric.

B. FUTURE WORK

This thesis provides numerous related opportunities for future work. First, the above results are only valid for the scenario provided in Chapter IV. Additional experimentation using new parameters is needed to determine if any universal conclusions can be made about the relationships presented here. For example, our simulation could be used to determine whether the range for the utilization factor is consistent across different scenarios or whether the range is dependent upon the individual parameters of the system (i.e., interarrival rate, number of machines and service rate)? The time to compute a schedule was negligible in this research. Yet to be analyzed is how the relationships change when the number of machines and jobs are very large and the time to compute a schedule is comparable to the time to service a job. This research then leads to the need for more study into how different scheduling algorithms affect the relationships presented here. Additionally, the effect of operating in different matrix heterogeneity categories needs to be further researched. Adding rescheduling to the simulation and analyzing the related costs and benefits also remains a consideration. Refining the simulation to more finely model the operating system ready and waiting queues would be beneficial. Finally, when the final QoS metric is defined for a particular environment in which MSHN will be used, the simulation could be altered to incorporate this metric, and any associated scheduling heuristics, to determine the cost tradeoff relationship for grouping jobs in that environment.

APPENDIX A: ACRONYMS AND SYMBOLS

AE	Application Emulator
C2	Command and Control
C4I	Command, Control, Communications, Computers, and Intelligence
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DES	Discrete Event Simulation
DoD	Department of Defense
ETC	Expected Time for Completion
G	Group Size
HC	Heterogeneous Computing
I/O	Input and/or Output
IDE	Integrated Development Environment
IID	Independent, Identically Distributed
J	Number of Jobs
JVM	JAVA Virtual Machine
M	Number of machines
MB	Megabyte
Mb/s	megabit/sec
MHz	Megahertz
ms	millisecond
MSHN	Management System for Heterogeneous Networks
NCCOSC	Naval Command, Control, and Ocean Surveillance Center
NFS	Network File System
NPS	Naval Postgraduate School
OLB	Opportunistic Load Balancing
OS	Operating System
QoS	Quality of Service
r	System Response Time
RMS	Resource Management System
RRD	Resource Requirements Database
RSS	Resource Status Server
s	Resource Service Time
SA	Scheduling Advisor
U(0,1)	A variate U that is uniformly distributed between 0 and 1.
VHM	Virtual Heterogeneous Machine
λ	Job Arrival Rate
μ	Resource Service Rate
ρ	Utilization Factor
τ	Job Interarrival Time

APPENDIX B: SOURCE CODE FOR SIMULATION MODEL

1. Source Code for class AssignContainer

```
//*************************************************************************
//   AssignContainer Class - MSHN_Sched
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*************************************************************************

/**
 * AssignContainer Class holds the data for job assignments determined from
 * the scheduler which can later be written to the job as it leaves the scheduler
 *
 * @author Major James Breitinger, USMC
 */
public class AssignContainer {

  /**
   * Holds the machine assignment
   */
  public int machine;

  /**
   * Holds the etc determined from the ETC matrix
   */
  public int etc;

  /**
   * Holds the tobeScheduled flag
   */
  public boolean flag;

  /**
   * Default Constructor for AssignContainer
   */
  public AssignContainer ( )
  {
    machine = 0;
    etc = 0;
    flag = true;
  }
```

```
  /**
   * Double int constructor and flag value for AssignContainer
   *
   * @param m machine number for assignment
   * @param e etc value from ETC Matrix and schedule
   * @param f flag value (true - needs to be scheduled, false - scheduled)
   */

  public AssignContainer ( int m, int e, boolean f )
  {
    machine = m;
    etc = e;
    flag = f;
  }

  /**
   * convert the AssignContainer into a string representation
   */
  public String toString()
  {
    return ("Machine assigned: "+machine+" ETC value: "+etc+" Flag: "+flag);
  }
}
```

2. Source Code for class ETCMatrix

```
//*****************************************************************************/
//   ETCMatrix Class - MSHN_SILK
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*****************************************************************************/

import com.threadtec.silk.random.*;
import java.util.Random;
import java.io.*;
import java.lang.Math;
import Simulation;

// ETCMatrix class derived from superclass QoSMatrix

/**
 * ETCMatrix Class builds an Expected Time to Compute matrix to be used for
 * scheduling by the SA Resource.  Can also display matrix and write to a file
 * (these lines of code are commented out by default).
 *
 * @author Major James Breitinger, USMC.
 *       Standard ETC adapted from code by LT Mike Niedert, USN.
 *       HiLo ETC adapted from code by Mr. Shoukat Ali, Purdue University.
```

```
*/
public final class ETCMatrix extends QoSMatrix {

  /**
   * file for Matrix output
   */
  private FileOutputStream output;

  /**
   * writer for Matrix output
   */
  private BufferedWriter out;

  /**
   * Constructor for class ETCMatrix
   *
   * @param rows number of rows in matrix
   * @param columns number of columns in matrix
   * @param seed seed value for random number generator
   */
  public ETCMatrix (int rows, int columns)
  {
    super(rows, columns);
  }

  /**
   * Constructor for class ETCMatrix
   *
   * @param rows number of rows in matrix
   * @param columns number of columns in matrix
   * @param seed seed value for random number generator
   * @param file file name for matrix output
   */
  public ETCMatrix (int rows, int columns, String file)
  {
    super(rows, columns);
    try {
      // Create a file object and an input stream object for the file
      output = new FileOutputStream(file);
      out = new BufferedWriter(new OutputStreamWriter(output));
    }
    catch(IOException e)        // File write exception
    {
      System.err.println(" Error writing to file" + e );
      System.exit(1);           // End the program
    }
  }
```

```
//*********************************
//GENERATE THE ETC MATRIX
//*********************************
/**
 * Generates a matrix of random integers between 1 and 100
 */
public int[][] generate()
{
  RandomStream ranETC = new RandomStream();
  int ETC[][] = new int[getJobs()][getMachines()];

  for(int job = 0; job < getJobs(); job++) {
    for(int machine = 0; machine < getMachines(); machine++) {
      ETC[job][machine] = (int)(ranETC.random()*100 + 1);
    }
  }
  return ETC;
}//END METHOD "generateETC"


//*********************************
// Generate a HiLo matrix
//*********************************
/**
 * Generates an inconsistent HiLo matrix
 */
public int[][] generateHiLo()
{
  double mean     = 100.0;  // mean ETC value along column
  double job_het  = 0.5;    // heterogeneity between jobs
  double mach_het = 0.1;    // heterogeneity between machines
  double j_alpha  = 1.0/(job_het*job_het);
  double j_beta   = mean/j_alpha;
  double j_std_dev = Math.sqrt(j_alpha*(j_beta*j_beta));

  Gamma gamJobMeanTime = new Gamma(mean,j_std_dev);
  double m_alpha      = 1.0/(mach_het*mach_het);
  double m_beta[]     = new double[Simulation.batchSize];
  double jobMean[]    = new double[Simulation.batchSize];
  double m_std_dev[]  = new double[Simulation.batchSize];

  for (int i = 0; i < Simulation.batchSize; i++) {
    jobMean[i]  = gamJobMeanTime.sample();
    m_beta[i]   = jobMean[i]/m_alpha;
    m_std_dev[i] = Math.sqrt(m_alpha*(m_beta[i]*m_beta[i]));
  }

  int ETC[][] = new int[getJobs()][getMachines()];
  for(int job = 0; job < getJobs(); job++) {
    /*System.out.println("Job #"+job+" Mean: "+(int)jobMean[job]+"  Std Dev : "
                +(int)m_std_dev[job]);*/  // for debugging
```

```
    Gamma gamETC    = new Gamma(jobMean[job],m_std_dev[job]);
    for(int machine = 0; machine < getMachines(); machine++) {
      ETC[job][machine] = (int)gamETC.sample();
    }
  }
  return ETC;
}//END METHOD "generateHiLo"


/**
 * Displays the ETC Matrix
 */
public void display(int matrix[][])
{
  System.out.print("        ");
  for(int column = 0; column < getMachines(); column++)
  {
    System.out.print("MACHINE #"+column+"   ");
  }
  System.out.println();
  for(int row = 0; row < getJobs(); row++)
  {
    System.out.print("JOB #"+row+"   ");
    for(int column = 0; column < getMachines(); column++)
    {
      System.out.print(matrix[row][column]+"         ");
    }
    System.out.println();
  }
  System.out.println();
}// end display


/**
 * Writes the ETC matrix to a file which is declared by user in
 * Simulation Class
 */
public void write(int matrix[][])
{
  try
  {
    out.write("        ");
    for(int column = 0; column < getMachines(); column++)
    {
      out.write("MACHINE #"+column+"   ");
    }
    out.newLine();
    for(int row = 0; row < getJobs(); row++)
    {
      out.write("JOB #"+row+"   ");
      for(int column = 0; column < getMachines(); column++)
```

```
            {
              out.write(matrix[row][column]+"          ");
            }
            out.newLine();
          }
          out.newLine();
          out.close();
          output.close();
        }
        catch(IOException e)        // File write exception
        {
          System.err.println(" Error writing to file" + e );
          System.exit(1);           // End the program
        }
      }
    }
```

3. Source Code for class InputFrame

```
//*****************************************************************************/
//   InputFrame Class - MSHN_SILK
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*****************************************************************************/
import java.awt.*;
import java.awt.event.*;

/**
 * A basic implementation of the Frame class
 *
 * @author Major James Breitinger, USMC
 */
public class InputFrame extends Frame implements ActionListener {

  private Label prompt1 = new Label( " Enter integer task batch size here:" );
  private TextField input1 = new TextField("10");
  private Label prompt2 =
      new Label( " Enter integer number of machines here:" );
  private TextField input2 = new TextField("10");
  private Label prompt3 =
      new Label ( " Enter a float task interarrival time here:");
  private TextField input3 = new TextField("6.0");
  private Button submit = new Button("Submit");
  private Panel one = new Panel();
  private Panel two = new Panel();
  private Panel three = new Panel();
  private Panel topPanel = new Panel();
```

```
    private Panel four = new Panel();

    /**
     * InputFrame constructor
     */
    public InputFrame() {

      super( "Input Simulation Parameters" );
      setBackground(Color.lightGray);
      setResizable( false );
      setLocation(150,250);

      one.setLayout(new GridLayout(1,2));
      one.add(prompt1);
      one.add(input1);

      two.setLayout(new GridLayout(1,2));
      two.add(prompt2);
      two.add(input2);

      three.setLayout(new GridLayout(1,2));
      three.add(prompt3);
      three.add(input3);

      four.add(submit, BorderLayout.CENTER);

      topPanel.setLayout(new GridLayout(3,1));
      topPanel.add(one);
      topPanel.add(two);
      topPanel.add(three);

      add(topPanel, BorderLayout.CENTER);
      add(four,BorderLayout.SOUTH);

      submit.addActionListener( this );
      pack();
      setVisible( true );

    }

    public void actionPerformed (ActionEvent e) {

      Simulation.batchSize = Integer.parseInt(input1.getText());
      Simulation.numMachines = Integer.parseInt(input2.getText());
      Job.arrive = (Double.valueOf(input3.getText())).doubleValue();

      dispose();

    }
}
```

4. Source Code for class Job

```
//*****************************************************************************/
//   Job Class - MSHN_SILK
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*****************************************************************************/

import com.threadtec.silk.*;
import com.threadtec.silk.random.*;
import com.threadtec.silk.statistics.*;
import com.threadtec.silk.util.*;

/**
 * Job Class drives the job entity thread as it traverses through the simulation
 *
 * @author Major James Breitinger, USMC
 */
public class Job extends Entity {

   /**
    * arrival time for each job
    */
   static Exponential expJobInterArrivalTime = new Exponential(arrive);

   /**
    * holds arrival time of job for recording time in Queue
    */
   double attArvTime;

   /**
    *  attribute of job for processing after scheduled
    */
   int machineAssign = 0;

   /**
    *  expected time to execute from ETC matrix
    */
   double executeTime = 30.0;

   /**
    * Job Class process method used to model Entity behavior.
    */
   public void process ( ) {

     // create next job arrival and assign arrival time to attribute
     create ( expJobInterArrivalTime.sample( ) );
     attArvTime = time;
```

```
        // wait for SA preload batching (Active SA resource entity will remove job from
        queue
        queue( queSA );

        // halt processing until activate method called from Active SA resource
        halt( );

        obsTimeWaitForSA.record( time - attArvTime );

        // wait for SA cycle (Active SA resource entity will remove job from queue
        queue( queBatch );

        // halt processing until activate method called from Active SA resource
        halt( );

        /*
        System.out.println("This jobs assignment is: "+machineAssign
                        + " for: "+executeTime); //for debugging
        */

        //wait for Machine cycle
        queue( queProcessor[machineAssign] );

        //double attArvTimeProc = time;

        while ( condition( resProcessor[machineAssign].getAvailability( ) == 0 ) );

        //obsTimeWaitForProcessor[machineAssign].record(time - attArvTimeProc);

        seize( resProcessor[machineAssign] );          // decrease availability
        dequeue( queProcessor[machineAssign] );        // remove Customer from queue


        // time delay: service time
        delay ( new Gamma(executeTime, 5.0).sample( ) );   // delay for service
        release ( resProcessor[machineAssign] );        // increase availability
        obsTimeInSystem.record(time - attArvTime);      // record time in system

        // recycle object
        dispose( );

    }
}
```

5. Source Code for class MSHN_Sched

```
//*****************************************************************************/
//   MSHN_Sched Class - MSHN_SILK
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*****************************************************************************/

import com.threadtec.silk.Silk;
import com.sun.java.swing.JApplet;

/**
Every Silk project requires this distinguished class which
instantiates a new <code>Silk</code> object for an application or applet.
*/

public class MSHN_Sched extends JApplet {

  Silk mshnSilk; // Declare a variable mshnSilk of class Silk

  /** This method is the starting point of execution when
     the program is run as an application. */

  public static void main ( String args[] ) {
    MSHN_Sched applicationSilk = new MSHN_Sched( );
    applicationSilk.start( );
  }

  /** When run as an applet, the browser calls this method
     when this class is first instantiated. */

  public void init( ) {
    Silk.setApplet( this );   // mark program as an applet
  }

  /** When run as an applet, this method is called each time
     the page containing the applet is revisited. */

  public void start ( ) {
    System.out.println ( "\nExecuting MSHN Schedule Simulation...\n" );
    mshnSilk  = new Silk( );
    mshnSilk.begin( );
  }

  /** When run as an applet, this method is called each time the
     page containing the applet is exited. */

  public void stop ( ) {
```

```
      com.threadtec.silk.util.CleanUp.purgeAll( );
      mshnSilk = null;

    }

}
```

6. Source Code for class QoSMatrix

```
//*****************************************************************************/
//   QoSMatrix abstract Class - MSHN_SILK
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*****************************************************************************/

/**
 * QoSMatrix Class is abstract for other specific matrix classes to extend and
 * inherit from
 */
public abstract class QoSMatrix {

  /**
   * the number of rows in the QoS matrix signifying the number of jobs
   */
  private int rows;

  /**
   * the number of columns in the QoS matrix signifying the number of machines
   */
  private int columns;

  /**
   * Constructor for QoSMatrix
   *
   * @param jobs number of rows in matrix
   * @param machines number of columns in matrix
   */
  public QoSMatrix (int jobs, int machines)
  {
    rows = jobs;
    columns = machines;
  }
```

```
  /**
   * Returns the number of jobs
   */
  public int getJobs()
  {
    return rows;
  }

  /**
   * Returns number of machines
   */
  public int getMachines()
  {
    return columns;
  }

  /**
   * Abstract method that must be implemented for each derived class
   * of QoSMatrix from which objects are instantiated.
   *
   * @return double subscripted array of int's representing matrix
   */
  abstract int[][] generate();

}
```

7. Source Code for class SA

```
//*****************************************************************************/
//   SA Class - MSHN_SILK
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*****************************************************************************/

import com.threadtec.silk.*;
import com.threadtec.silk.random.*;

/**
 * SA Class is an active resource that handles the scheduling process
 *
 * @author Major James Breitinger, USMC
 */
public class SA extends Entity {

  AssignContainer sched[] = new AssignContainer[batchSize];

  /**
```

```
 * Job Class entities being scheduled
 */
static Job entJob = null;

double attArrivalTime;

/**
 * SA Class process method used to model Entity behavior.
 */
public void process ( ) {

  while ( true )
  {
    // build ETC Matrix for this batch
    ETCMatrix hiloMatrix = new ETCMatrix(batchSize,numMachines);
    //ETCMatrix hiloMatrix = new ETCMatrix(batchSize,numMachines,"matrices.dat");
    int ETC[][] = hiloMatrix.generateHiLo();
    //hiloMatrix.write(ETC);   // for debugging
    //hiloMatrix.display(ETC); // for debugging

    // start SA operation when SA queue exceeds specified batch size
    while( condition ( queSA.getLength( ) < batchSize ) );

    // *** delay to load SA
    seize( resSA );

    // record batchsize for this cycle
    obsBatchSize.record(batchSize);

    // *** remove jobs from SA queue
    for( int i = 0; i < batchSize; i++ ) {
      entJob = (Job)queSA.remove(1);
      entJob.activate( );
    }

    attArrivalTime = time;

    // *** delay for SA cycle to calculate schedule

    System.gc();
    delay(maxminSchedule(sched,ETC) );

    obsTimetoSched.record(time - attArrivalTime);

    // *** remove jobs from SA (restart process for Job)
    for( int i = 0; i < batchSize; i++ ) {
      entJob = (Job)queBatch.remove(1);
      entJob.machineAssign = sched[i].machine;
      entJob.executeTime = (double)(sched[i].etc);
      entJob.activate( );
    }
```

```
    //System.out.println("SA is done!"); //for debugging

    // *** start next cycle
    release( resSA );
  }
}

/**
 * Schedules each job to a machine based upon the min value in
 * the given matrix for each job
 *
 * @return  array of <code>AssignContainers</code> representing job assignments
 */
public double minSchedule(AssignContainer[] sched, int[][] matrix)
{
  double tempTime = (double)System.currentTimeMillis();

  for( int j = 0; j < matrix.length; j++ ) {
    int etc = matrix[j][0];
    int machine = 0;
    for (int m = 1; m < matrix[j].length; m++) {
      if (matrix[j][m] < etc) {
        etc = matrix[j][m];
        machine = m;
      }
    }
    sched[j] = new AssignContainer(machine, etc, false);
  }

  double temptime2 = (double)System.currentTimeMillis();
  double CPUtime = (temptime2 - tempTime);
  return (double)CPUtime;
} // end minSchedule


/**
 * Schedules each job to a machine based upon the min value in each row of
 * the given matrix for each job and then assigns the maximum of those minimums
 * This algorithm in a greedy MAXMIN algorithm
 *
 * @return  array of <code>AssignContainers</code> representing job assignments
 */
public double maxminSchedule(AssignContainer[] sched, int[][] matrix)
{
  AssignContainer intermediate[] = new AssignContainer[matrix.length];
  int[][] temp = new int[matrix.length][matrix[0].length];
  int etc = 0;
  int machine = 0;
  int assign = 0;

  double tempTime = (double)System.currentTimeMillis();
```

```
// copy the matrix for revision
for( int a = 0; a < matrix.length; a++ ) { // iterate throught the rows
  for( int b = 0; b < matrix[a].length; b++ ) { // iterate through the columns
    temp[a][b] = matrix[a][b];
  }
}

// build default sched[] array
for(int k = 0; k < matrix.length; k++) {
  sched[k] = new AssignContainer();
}

// begin scheduling
for( int s = 0; s < matrix.length; s++ ) { // do this for each job to be assigned
  for( int j = 0; j < matrix.length; j++ ) { // iterate through the rows
    if(sched[j].flag) { // only do tasks not scheduled
      etc = temp[j][0];
      machine = 0;
      for (int m = 1; m < matrix[j].length; m++) { //iterate through the columns
        if (temp[j][m] < etc) {  // find the min value for the row
          etc = temp[j][m];
          machine = m;
        }
      }
      intermediate[j] = new AssignContainer(machine, etc, true);
    }
  }

  // iterate through the intermediate and find the max
  etc = 0;
  for(int i = 0; i < intermediate.length; i++) {
    if(sched[i].flag && intermediate[i].etc > etc) {
      etc = intermediate[i].etc;
      machine = intermediate[i].machine;
      assign = i;
    }
  }
  sched[assign] = new AssignContainer(machine,matrix[assign][machine],false);

  // revise the matrix with the new values from the assignment
  for (int k = 0; k < matrix.length; k++) {
    temp[k][machine] = temp[k][machine] + etc;
  }
}

double temptime2 = (double)System.currentTimeMillis();
double CPUtime = (temptime2 - tempTime);

//System.out.println("The time to schedule was: "+CPUtime+"ms"); // for debugging
```

```
    return (double)CPUtime;
  } // end maxminSchedule

}
```

8. Source Code for class Simulation

```
//*****************************************************************************/
//   Simulation Class - MSHN_Sched
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*****************************************************************************/
//
// Entity Objects
//   Job
//   SA
//
// Resource Objects
//   SA - Scheduling Advisor (Active Resource)
//   Processors[] - array of machines
//
//******* This class references classes included in the following files ********/

import com.threadtec.silk.*;
import com.threadtec.silk.gui.*;
import com.threadtec.silk.random.*;
import com.threadtec.silk.statistics.*;
import java.io.*;

/**
 * Simulation Class starts the simulation, loads the Control Console, and declares
 * the global variables needed in the simulation
 *
 * @author Major James Breitinger, USMC
 */

public class Simulation extends Silk {

   // *** Declarations performed here are global to all Entity classes

   /**
    * The size of the pool of jobs to be scheduled
    */
   public static int          batchSize       = 10;

   /**
    * The number of Machines in the heterogeneous environment where the pool
```

```
    * of jobs are to be scheduled
    */
   public static int          numMachines     = 15;

   /**
    * The mean interarrival time for jobs arriving at the system
    */
   public static double       arrive          = 67.0;

   /**
    * The container for the pool of jobs waiting to be scheduled
    */
   public static Queue        queBatch        = new Queue("Batch Container");

   /**
    * The Queue holding the arriving jobs waiting to be scheduled
    */
   public static Queue        queSA           = new Queue("SA Queue");

   /**
    * The array of queues holding jobs waiting to be processed on their assigned
    * machines
    */
   public static Queue        queProcessor[];

   /**
    * The Scheduling Advisor resource needed to represent the scheduler
    */
   public static Resource     resSA           = new Resource("SA");

   /**
    * The array of machines for the jobs to be processed
    */
   public static Resource     resProcessor[];

   // Observational Statistics
   public static Observational  obsTimeWaitForSA = new Observational("Wait time for
SA"),
                        obsBatchSize    = new Observational("Batch Size"),
                        obsTimetoSched  = new Observational("Time to Schedule"),
                        obsTimeInSystem = new Observational("Time in System");
                        //obsTimeWaitForProcessor[];

   public static TimeDependent  tdSAUtil       = new
TimeDependent(resSA.numBusy,"SA Utilization"),
                        tdSAQueueLength = new TimeDependent(queSA.length,"SA
Queue Length");

   //public static TimeDependent  tdMachUtil[],
   //                     tdMachQueueLength[];
```

```
    /**
     * Initializes variables needed for the simulation
     */
    public void init ( ) {

      queProcessor     = new Queue[numMachines];
      resProcessor     = new Resource[numMachines];
      //obsTimeWaitForProcessor = new Observational[numMachines];
      //tdMachUtil       = new TimeDependent[numMachines];
      //tdMachQueueLength = new TimeDependent[numMachines];

      for(int i = 0; i < numMachines; i++) {
        queProcessor[i] = new Queue("Queue " + i);
        resProcessor[i] = new Resource("Resource " + i);
        //obsTimeWaitForProcessor[i] = new Observational("Wait time for Processor #"+i);
        //tdMachUtil[i]   = new TimeDependent
        //   (resProcessor[i].numBusy,"Processor #"+i+" Utilization");
        //tdMachQueueLength[i] = new TimeDependent
        //   (queProcessor[i].length,"Processor #"+i+" Queue Length");
      }

      SA entSA;
      Job entFirstJob;
      StatManager entStatManager;

      entFirstJob = (Job)newEntity( Job.class );     // create first instance of Job object
      entFirstJob.start( 0.0 );                      // schedule initial arrival

      entSA = (SA)newEntity( SA.class );             // create instance of SA object
      entSA.start( 0.0 );

      entStatManager = (StatManager)newEntity( StatManager.class );
      entStatManager.start( 10000.0 );

    }

    /**
     * Starts the simulation and initializes input frame
     */
    public void run ( ) {

      setReplications( 30 );
      setRunLength( 110000.);

      InputFrame ijf = new InputFrame( );

      setControlConsole( true );

    }

}
```

9. Source Code for class MSHN_Sched

```
//*****************************************************************************/
//   StatManager Class - MSHN_SILK
//
//   Major James Breitinger, USMC and Dr. Taylor Kidd
//   Naval Postgraduate School 1999
//   http://www.mshn.org
//*****************************************************************************/

import com.threadtec.silk.*;
import com.threadtec.silk.statistics.*;

/**
 * StatManager Class resets the Simulation Statistics to allow the
 * Simulation to populate itself for realistic testing of a fully operating
 * system without "startup" latency
 *
 * @author Major James Breitinger, USMC
 */
public class StatManager extends Entity {

  /**
   * StatManager Class process method used for Entity behavior.
   */
  public void process ( )
  {
    reSetStats( );
    dispose( );
  }
}
```

APPENDIX C: JAVA DOCUMENTATION FOR SIMULATION MODEL

1. MSHN_Sched API in HTML format

MSHN_Sched API

MSHN_Sched Application Programming Interface

User's Guide

How the API Is Organized

There are two levels to the API:

- All Classes (within a package)
- This Class (selected class).

Level 1 - All Classes

This level provides links to the classes and interfaces in a given package. There are three catogories in the listing:

- Interfaces
- Classes
- Exceptions

Level 2 - This Class/Interface

This level begins with an index, followed by the detailed API. There are three categories at the class level.

- Variables
- Constructors
- Methods

A category is omitted when a class has no applicable entries.
Within these categories there is additional color coding as follows:

- Instance Variables
- Static Variables
- Constructors
- Instance Methods
- Static Methods

How to Locate Items

- To Browse A Class
  - Select a class from the list of All Classes. This list is the home page for the the MSHN_Sched API.
- To Locate a Class
  - Use the searchable index tool.

    - Select the class from the alphabetical index.
- To Browse a Class
    - Use the Next/Previous anchors to browse alphabetically.
    - Or, traverse the links within the class.
- To Locate a Method
    - Use the searchable index tool.
    - Or, scroll through the alphabetical class index to locate a method.

A Closer Look at the Class-Level API

Take a look at class MSHN_Sched. The navigational anchors are at the top. This is followed by the fully qualified class name and a representation of its position in the class hierarchy.

The next entries are links to the superclass and the interfaces, if any. This is followed by a description of the class, taken from the class comment. Notice how the programmer has embedded some code samples using html tags.

The author also chose to include a See Also entry to another class. Following the class-level entries for See Also, Version, and Author, the index begins.

The Index

Each class/interface begins with an index of its variables, constructors and methods, sorted alphabetically. The entry consists of the declaration and short description. The description is the first sentence of the doc comment for that item. The index entries are linked to their corresponding entries in the application programming inteface which immediately follows.

The Detailed API

The index is followed by the complete API for each entry. Within the three categories: Variables, Constructors, and Methods, the entries are presented in the order they appear in the source. This is done to preserve the logical groupings established by the programmer.

Where Are All the Links in the API?

- There are links in the class type of every method and variable definition.
- At the top of each class/interface there is a drawing of the tree structure down to the current class/interface, in which each superclass is a link.
- Every method contains a list of exceptions that it may throw. These are linked to the appropriate class.
- The superclass and interface references at the beginning of the class are links.
- Every See Also is a link.
- When a method overrides a method in the superclass, the API has the entry "Overrides: foo in class bar." Both foo (the method name) and bar (the class name) are links.

---

2. Class Index in HTML format

API User's Guide Class Hierarchy Index

Class Index

- AssignContainer
- ETCMatrix
- InputFrame
- Job
- MSHN_Sched
- QoSMatrix
- SA
- Simulation
- StatManager

3. Class Hierarchy in HTML format

All Classes Index

Class Hierarchy

- class java.lang.Object
    - class AssignContainer
    - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
        - class java.awt.Container
            - class java.awt.Panel
                - class java.applet.Applet
                    - class com.sun.java.swing.JApplet (implements com.sun.java.accessibility.Accessible, com.sun.java.swing.RootPaneContainer)
                        - class MSHN_Sched
            - class java.awt.Window
                - class java.awt.Frame (implements java.awt.MenuContainer)
                    - class InputFrame (implements java.awt.event.ActionListener)
    - class QoSMatrix
        - class ETCMatrix
    - class com.threadtec.silk.Silk (implements java.lang.Runnable)
        - class Simulation
            - class com.threadtec.silk.Entity
                - class Job
                - class SA
                - class StatManager

4. AssignContainer Class in HTML format

Class AssignContainer

```
java.lang.Object
   |
   +----AssignContainer
```

---

public class **AssignContainer**

extends Object AssignContainer Class holds the data for job assignments determined from the scheduler which can later be written to the job as it leaves the scheduler

Author:

Major James Breitinger, USMC

Variable Index

- **etc**

  Holds the etc determined from the ETC matrix

- **flag**

  Holds the tobeScheduled flag

- **machine**

  Holds the machine assignment

Constructor Index

- **AssignContainer**()

  Default Constructor for AssignContainer

- **AssignContainer**(int, int, boolean)

  Double int constructor and flag value for AssignContainer

Method Index

- toString()

  convert the AssignContainer into a string representation

Variables

- **machine**

```
public int machine
```

  Holds the machine assignment

- **etc**

```
public int etc
```

  Holds the etc determined from the ETC matrix

- **flag**

```
public boolean flag
```

  Holds the tobeScheduled flag

Constructors

- **AssignContainer**

```
public AssignContainer()
```

  Default Constructor for AssignContainer

- **AssignContainer**

```
public AssignContainer(int m,
                       int e,
                       boolean f)
```

  Double int constructor and flag value for AssignContainer

Parameters:

  m - machine number for assignment

  e - etc value from ETC Matrix and schedule

  f - flag value (true - needs to be scheduled, false - scheduled)

Methods

- **toString**

```
public String toString()
```

convert the AssignContainer into a string representation

Overrides:

toString in class Object

5. ETCMatrix Class in HTML format

Class ETCMatrix

```
java.lang.Object
   |
   +----QoSMatrix
           |
           +----ETCMatrix
```

---

public final class **ETCMatrix**

extends QoSMatrix ETCMatrix Class builds an Expected Time to Compute matrix to be used for scheduling by the SA Resource. Can also display matrix and write to a file (these lines of code are commented out by default).

Author:

Major James Breitinger, USMC.

Standard ETC adapted from code by LT Mike Niedert, USN.

HiLo ETC adapted from code by Mr. Shoukat Ali, Purdue University.

Constructor Index

- **ETCMatrix**(int, int)

  Constructor for class ETCMatrix

- **ETCMatrix**(int, int, String)

  Constructor for class ETCMatrix

Method Index

- **display**(int[][])

  Displays the ETC Matrix

- **generate**()

  Generates a matrix of random integers between 1 and 100

- **generateHiLo**()

  Generates an inconsistent HiLo matrix

- **write**(int[][])

  Writes the ETC matrix to a file which is declared by user in Simulation Class

Constructors

- **ETCMatrix**

```
public ETCMatrix(int rows,
                 int columns)
```

  Constructor for class ETCMatrix

Parameters:

  rows - number of rows in matrix

  columns - number of columns in matrix

  seed - seed value for random number generator

- **ETCMatrix**

```
public ETCMatrix(int rows,
           int columns,
           String file)
```

  Constructor for class ETCMatrix

Parameters:

  rows - number of rows in matrix

  columns - number of columns in matrix

  seed - seed value for random number generator

  file - file name for matrix output

Methods

- **generate**

```
public int[][] generate()
```

  Generates a matrix of random integers between 1 and 100

Overrides:

generate in class QoSMatrix

- **generateHiLo**

```
public int[][] generateHiLo()
```

Generates an inconsistent HiLo matrix

- **display**

```
public void display(int matrix[][])
```

Displays the ETC Matrix

- **write**

```
public void write(int matrix[][])
```

Writes the ETC matrix to a file which is declared by user in Simulation Class

6. InputFrame Class in HTML format

Class InputFrame

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----InputFrame
```

---

public class **InputFrame**

extends Frame

implements ActionListener A basic implementation of the Frame class

Author:

Major James Breitinger, USMC

Constructor Index

- **InputFrame**()

  InputFrame constructor

Method Index

- **actionPerformed**(ActionEvent)

Constructors

- **InputFrame**

```
public InputFrame()
```

InputFrame constructor

Methods

- **actionPerformed**

```
public void actionPerformed(ActionEvent e)
```

7. Job Class in HTML format

Class Job

```
java.lang.Object
   |
   +----com.threadtec.silk.Silk
           |
           +----Simulation
                   |
                   +----com.threadtec.silk.Entity
                           |
                           +----Job
```

---

public class **Job**

extends Entity Job Class drives the job entity thread as it traverses through the simulation

Author:

Major James Breitinger, USMC

Variable Index

- **arrive**

  default mean arrival rate is 6.0 time units

Constructor Index

- Job()

Method Index

- **process**()

Variables

Job Class process method used to model Entity behavior.

- **arrive**

```
public static double arrive
```

default mean arrival rate is 6.0 time units

Constructors

- **Job**

Methods

```
public Job()
```

- **process**

```
public void process()
```

Job Class process method used to model Entity behavior.

Overrides:

process in class Entity

8. MSHN_Sched Class in HTML format

Class MSHN_Sched

```
java.lang.Object
    |
    +----java.awt.Component
              |
              +----java.awt.Container
                        |
                        +----java.awt.Panel
                                  |
                                  +----java.applet.Applet
                                            |
                                            +----com.sun.java.swing.JApplet
                                                      |
                                                      +----MSHN_Sched
```

---

public class **MSHN_Sched**

extends JApplet Every Silk project requires this distinguished class which instantiates a new `Silk` object for an application or applet.

Constructor Index

- **MSHN_Sched**()

Method Index

- **init**()

  When run as an applet, the browser calls this method when this class is first instantiated.

- **main**(String[])

  This method is the starting point of execution when the program is run as an application.

- **start**()

  When run as an applet, this method is called each time the page containing the applet is revisited.

- stop()

  When run as an applet, this method is called each time the page containing the applet is exited.

Constructors

- **MSHN_Sched**

```
public MSHN_Sched()
```

Methods

- **main**

```
public static void main(String args[])
```

  This method is the starting point of execution when the program is run as an application.

- **init**

```
public void init()
```

  When run as an applet, the browser calls this method when this class is first instantiated.

Overrides:

  init in class Applet

- **start**

```
public void start()
```

  When run as an applet, this method is called each time the page containing the applet is revisited.

Overrides:

  start in class Applet

- **stop**

```
public void stop()
```

  When run as an applet, this method is called each time the page containing the applet is exited.

Overrides:

  stop in class Applet

9. QoSMatrix Class in HTML format

Class QoSMatrix

```
java.lang.Object
   |
   +----QoSMatrix
```

---

public abstract class **QoSMatrix**

extends Object QoSMatrix Class is abstract for other specific matrix classes to extend and inherit from

---

Constructor Index

- **QoSMatrix**(int, int)

  Constructor for QoSMatrix

Method Index

- **getJobs**()

  Returns the number of jobs

- **getMachines**()

  Returns number of machines

Constructors

- **QoSMatrix**

```
public QoSMatrix(int jobs,
                 int machines)
```

  Constructor for QoSMatrix

Parameters:

  jobs - number of rows in matrix

  machines - number of columns in matrix

Methods

- **getJobs**

```
public int getJobs()
```

Returns the number of jobs

- **getMachines**

```
public int getMachines()
```

Returns number of machines

10. SA Class in HTML format

Class SA

```
java.lang.Object
   |
   +----com.threadtec.silk.Silk
           |
           +----Simulation
                   |
                   +----com.threadtec.silk.Entity
                           |
                           +----SA
```

---

public class **SA**

extends Entity SA Class is an active resource that handles the scheduling process

Author:

Major James Breitinger, USMC

Constructor Index

- SA()

Method Index

- **maxminSchedule**(AssignContainer[], int[][])

  Schedules each job to a machine based upon the min value in each row of the given matrix for each job and then assigns the maximum of those minimums This algorithm in a greedy MAXMIN algorithm

- **minSchedule**(AssignContainer[], int[][])

  Schedules each job to a machine based upon the min value in the given matrix for each job

- **process**()

  SA Class process method used to model Entity behavior.

Constructors

- **SA**

```
public SA()
```

Methods

- **process**

```
public void process()
```

SA Class process method used to model Entity behavior.

Overrides:

process in class Entity

- **minSchedule**

```
public double minSchedule(AssignContainer sched[],
                          int matrix[][])
```

Schedules each job to a machine based upon the min value in the given matrix for each job

Returns:

array of `AssignContainers` representing job assignments

- **maxminSchedule**

```
public double maxminSchedule(AssignContainer sched[],
                             int matrix[][])
```

Schedules each job to a machine based upon the min value in each row of the given matrix for each job and then assigns the maximum of those minimums This algorithm in a greedy MAXMIN algorithm

Returns:

array of `AssignContainers` representing job assignments

11. Simulation Class in HTML format

Class Simulation

```
java.lang.Object
   |
   +----com.threadtec.silk.Silk
           |
           +----Simulation
```

---

public class **Simulation**

extends Silk Simulation Class starts the simulation, loads the Control Console, and declares the global variables needed in the simulation

Author:

Major James Breitinger, USMC

Variable Index

- **batchSize**

  The size of the pool of jobs to be scheduled

- **numMachines**

  The number of Machines in the heterogeneous environment where the pool of jobs are to be scheduled

- **obsBatchSize**
- **obsTimeInSystem**
- **obsTimetoSched**
- **obsTimeWaitForSA**
- **queBatch**

  The container for the pool of jobs waiting to be scheduled

- **queProcessor**

  The array of queues holding jobs waiting to be processed on their assigned machines

- **queSA**

  The Queue holding the arriving jobs waiting to be scheduled

- **resProcessor**

  The array of machines for the jobs to be processed

- **resSA**

  The Scheduling Advisor resource needed to represent the scheduler

- **tdSAQueueLength**
- **tdSAUtil**

Constructor Index

- **Simulation**()

Method Index

- **init**()

  Initializes variables needed for the simulation

- **run**()

  Starts the simulation and initializes input frame

Variables

- **batchSize**

```
public static int batchSize
```

  The size of the pool of jobs to be scheduled

- **numMachines**

```
public static int numMachines
```

  The number of Machines in the heterogeneous environment where the pool of jobs are to be scheduled

- **queBatch**

```
public static Queue queBatch
```

  The container for the pool of jobs waiting to be scheduled

- **queSA**

```
public static Queue queSA
```

The Queue holding the arriving jobs waiting to be scheduled

- **queProcessor**

```
public static Queue queProcessor[]
```

The array of queues holding jobs waiting to be processed on their assigned machines

- **resSA**

```
public static Resource resSA
```

The Scheduling Advisor resource needed to represent the scheduler

- **resProcessor**

```
public static Resource resProcessor[]
```

The array of machines for the jobs to be processed

- **obsTimeWaitForSA**

```
public static Observational obsTimeWaitForSA
```

- **obsBatchSize**

```
public static Observational obsBatchSize
```

- **obsTimetoSched**

```
public static Observational obsTimetoSched
```

- **obsTimeInSystem**

```
public static Observational obsTimeInSystem
```

- **tdSAUtil**

```
public static TimeDependent tdSAUtil
```

- **tdSAQueueLength**

```
public static TimeDependent tdSAQueueLength
```

Constructors

- **Simulation**

```
public Simulation()
```

Methods

- **init**

```
public void init()
```

Initializes variables needed for the simulation

Overrides:

init in class Silk

- **run**

```
public void run()
```

Starts the simulation and initializes input frame

Overrides:

run in class Silk

12. StatManager Class in HTML format

Class StatManager

```
java.lang.Object
   |
   +----com.threadtec.silk.Silk
          |
          +----Simulation
                  |
                  +----com.threadtec.silk.Entity
                          |
                          +----StatManager
```

---

public class **StatManager**

extends Entity StatManager Class resets the Simulation Statistics to allow the Simulation to populate itself for realistic testing of a fully operating system without "startup" latency

Author:

Major James Breitinger, USMC

Constructor Index

- StatManager()

Method Index

- process()

  StatManager Class process method used for Entity behavior.

Constructors

- **StatManager**

```
public StatManager()
```

Methods

- **process**

```
public void process()
```

StatManager Class process method used for Entity behavior.

Overrides:

process in class Entity

13. Index of all Fields and Methods in HTML format

Index of all Fields and Methods

A

actionPerformed(ActionEvent). Method in class InputFrame

arrive. Static variable in class Job

default mean arrival rate is 6.0 time units

AssignContainer(). Constructor for class AssignContainer

Default Constructor for AssignContainer

AssignContainer(int, int, boolean). Constructor for class AssignContainer

Double int constructor and flag value for AssignContainer

B

batchSize. Static variable in class Simulation

The size of the pool of jobs to be scheduled

D

display(int[][]). Method in class ETCMatrix

Displays the ETC Matrix

E

etc. Variable in class AssignContainer

Holds the etc determined from the ETC matrix

ETCMatrix(int, int). Constructor for class ETCMatrix

Constructor for class ETCMatrix

ETCMatrix(int, int, String). Constructor for class ETCMatrix

Constructor for class ETCMatrix

F

flag. Variable in class AssignContainer

Holds the tobeScheduled flag

G

generate(). Method in class ETCMatrix

Generates a matrix of random integers between 1 and 100

generateHiLo(). Method in class ETCMatrix

Generates an inconsistent HiLo matrix

getJobs(). Method in class QoSMatrix

Returns the number of jobs

getMachines(). Method in class QoSMatrix

Returns number of machines

I

init(). Method in class MSHN_Sched

When run as an applet, the browser calls this method when this class is first instantiated.

init(). Method in class Simulation

Initializes variables needed for the simulation

InputFrame(). Constructor for class InputFrame

InputFrame constructor

J

Job(). Constructor for class Job

M

machine. Variable in class AssignContainer

Holds the machine assignment

main(String[]). Static method in class MSHN_Sched

This method is the starting point of execution when the program is run as an application.

maxminSchedule(AssignContainer[], int[][]). Method in class SA

Schedules each job to a machine based upon the min value in each row of the given matrix for each job and then assigns the maximum of those minimums This algorithm in a greedy MAXMIN algorithm

minSchedule(AssignContainer[], int[][]). Method in class SA

Schedules each job to a machine based upon the min value in the given matrix for each job

MSHN_Sched(). Constructor for class MSHN_Sched

N

numMachines. Static variable in class Simulation

The number of Machines in the heterogeneous environment where the pool of jobs are to be scheduled

O

obsBatchSize. Static variable in class Simulation

P

Q

R

resProcessor. Static variable in class Simulation

The array of machines for the jobs to be processed

resSA. Static variable in class Simulation

The Scheduling Advisor resource needed to represent the scheduler

run(). Method in class Simulation

Starts the simulation and initializes input frame

S

SA(). Constructor for class SA

Simulation(). Constructor for class Simulation

start(). Method in class MSHN_Sched

When run as an applet, this method is called each time the page containing the applet is revisited.

StatManager(). Constructor for class StatManager

stop(). Method in class MSHN_Sched

When run as an applet, this method is called each time the page containing the applet is exited.

T

tdSAQueueLength. Static variable in class Simulation

tdSAUtil. Static variable in class Simulation

toString(). Method in class AssignContainer

convert the AssignContainer into a string representation

W

write(int[][]). Method in class ETCMatrix

Writes the ETC matrix to a file which is declared by user in Simulation Class

APPENDIX D: HOW TO RUN THE SIMULATION

In order to run the simulation, your environment must first be set properly. Because the Silk simulation tool is designed for JAVA 1.1.8 and earlier, the primary reason for setting the environment is to have the JAVA CLASSPATH point to where the `Silk.jar` and `swingall103.jar` files are located. I recommend both of these files be located in a directory called `SILK`. If an Integrated Development Environment (IDE) is used to modify the simulation classes, then the CLASSPATH can be set from within the environment tool and the simulation run from within the IDE. The KAWA 3.22 development tool for Windows was used to write and compile the JAVA classes for this thesis.

Silk can also be configured to work across a network through the use of a web browser. This involves the additional configuration of the browser. Unless the simulation is at some point changed to include JAVA Swing animation, I do not recommend using a browser. However, additional information on this topic can be found at http://www.threadtec.com/demos.html.

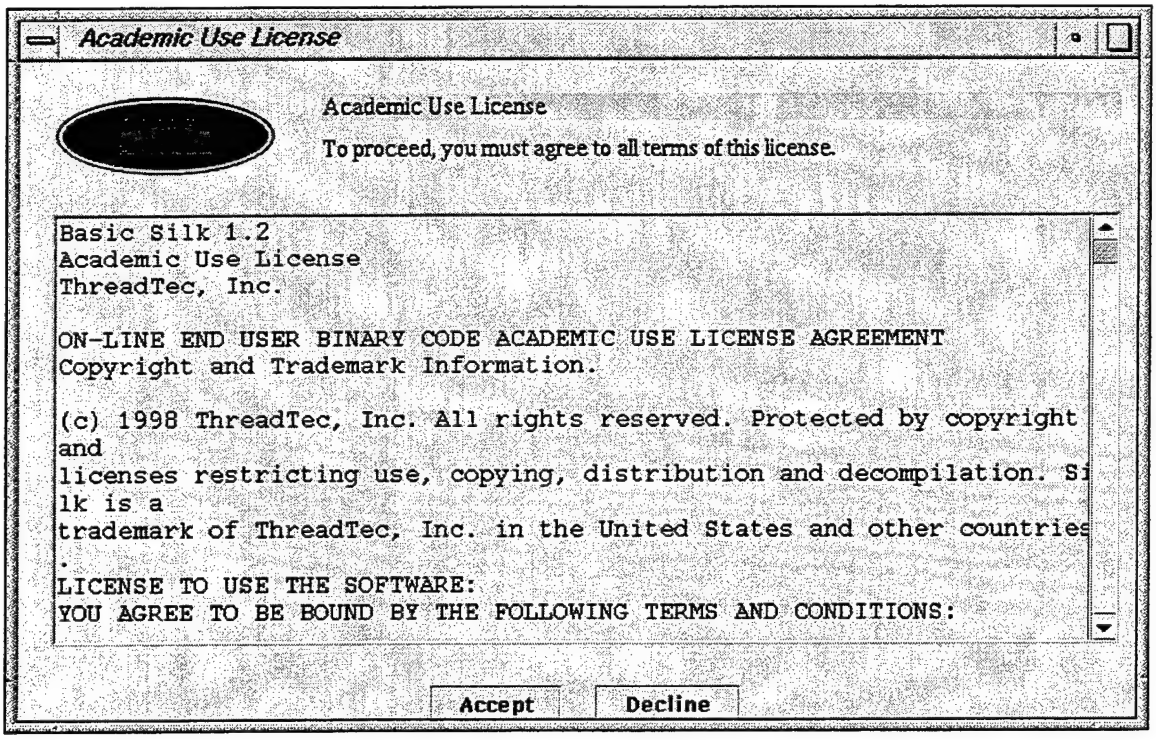
The simulation can be started from either an IDE or from a command prompt. You launch the program from within the directory where the simulation "CLASS" files are located by typing `java MSHN_Sched`. For example, on my machine at NPS, I would type `D:\thesis\mshn_silk\java MSHN_Sched`.

Because Windows NT does not provide resolution down to the millisecond, I ran the simulation on *caesar*, our Silicon Graphics Challenge L. Setting the environment on this machine was slightly different than above. Before running the simulation, first copy the simulation CLASS files to a directory on *caesar* and then copy the `Silk.jar` and `swingall103.jar` files to a directory named `Silk`. Then alter the `.login` file in the "HOME" directory by adding the following lines:

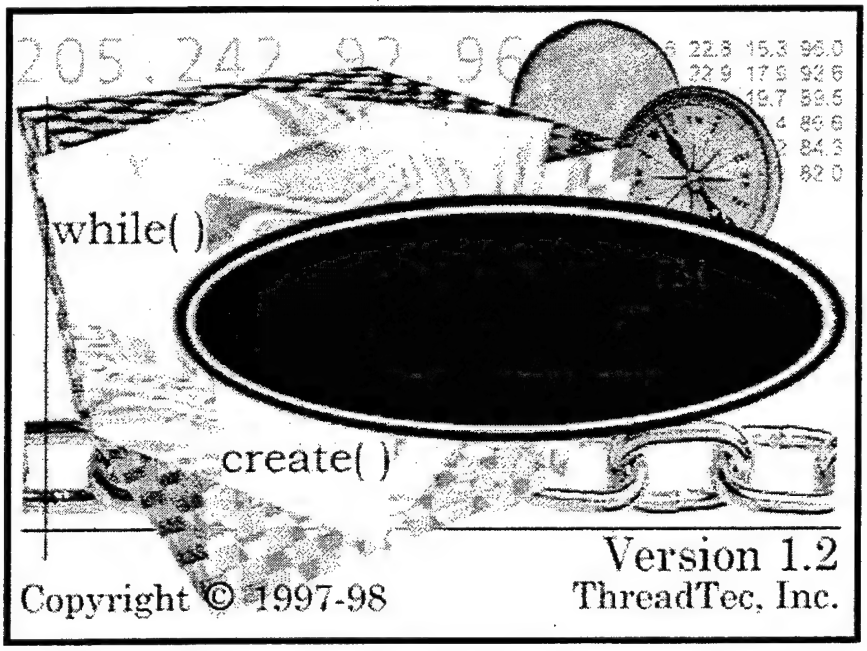
```
setenv JAVA_HOME /usr/java
setenv CLASSPATH .:$HOME/Silk/Silk.jar:$HOME/Silk/swingall103.jar
```

Once completed, the program is started the same as under a Windows system described above.

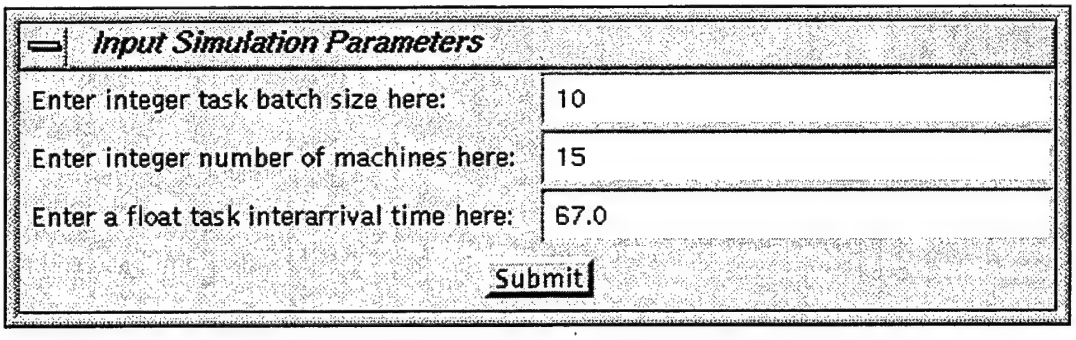
The first screen to appear as the simulation starts is the Silk academic use license. Select "ACCEPT" and this screen disappears.



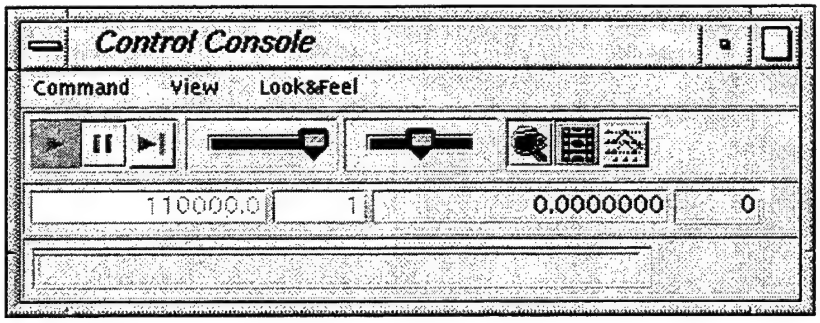
Next the Silk logo appears and vanishes after a few seconds.



The next screens to appear are the Input screen and the Control Console.



The input screen is used to input the system parameters for the simulation. The "Submit" button must be clicked prior to starting the simulation from the control console.



Altering the number in the first window of the control console changes the amount of simulation time for the run and the number of runs completed can be altered by changing the second window. The simulation is started by either selecting RUN from the COMMAND drop-down menu or by clicking the right-facing arrow on the console.

When the run is complete, the statistical results are automatically displayed in the Summary Output window. Altering the variables to be watched in the simulation model code changes the types of data recorded in the summary output. Refer to the Silk documentation to make these changes. In order to record the results from the Summary Output screen, you must first cut-and-paste the data to a text file.

Because the simulation is built entirely from JAVA code, a knowledge of JAVA programming, the Silk documentation included with the Silk tool, and the documentation in APPENDIX B are all that is necessary to alter the objects in or output from the simulation.

LIST OF REFERENCES

[ARMS97] Armstrong, Robert K."*Investigation of Effect of Different Run-Time Distributions on SmartNet Performance.*" Master's Thesis. Naval Postgraduate School, Monterey, California, September 1997.

[ARMS98] Armstrong, Robert, Debra Hensgen, and Taylor Kidd, "*The Relative Performance of Various Mapping Algorithms is independent of Sizable Variances in Runtime Predictions.*" Proceedings HCW'98, March 1998.

[BRAU98] Braun, Tracy D., Muthucumaru Maheswaran, Howard Jay Siegel, Noah Beck, Ladislau Boloni, Albert Reuther, James Robertson, Mitchell Theys, and Bin Yao. "*A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems.*" Workshop on Advances in Parallel and Distributed Systems, October 10, 1998.

[BRAU99] Braun, Tracy D., Howard Jay Siegel, Noah Beck, Ladislau Boloni, Muthucumaru Maheswaran, Albert Reuther, James Robertson, Mitchell Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. "*A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems,*" Proceedings HCW'99, April 1999.

[FLAN97] Flanagan, David. JAVA in a Nutshell, 2nd Edition. Sebastopol, CA: O'Reilly, 1997.

[HAMM64] Hammersley, J.M. and D.C. Handscomb. Monte Carlo Methods. London: Chapman & Hall, 1964.

[HENS99] Hensgen, D., T. Kidd, D. St. John, M. Schnaidt, H. J. Siegel, T. Braun, J. Kim, S. Ali, C. Irvine, T. Levin V. Prasanna, P. Bhat, R. Freund, and M. Godfrey. "*An Overview of MSHN: The Management System for Heterogeneous Networks,*" 8th IEEE Workshop on Heterogeneous Computing Systems (HCW'99), San Juan, Puerto Rico, pp. 184-198, April 1999.

[IBAR77] Ibarra, Oscar H. and Chul E. Kim. "*Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors.*" Journal of the ACM, 24, No. 2, April 1977, 280-289.

[JAIN91] Jain, Raj. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. New York: Wiley, 1991.

[JANA96] Janakiraman, Mathruboothan. "*Simulation Results for Heuristic Algorithms for Scheduling Precedence-Related Tasks in Heterogeneous Environments.*" Master's Thesis. University of Cincinnati, 1996.

[KIDD96] Kidd, Taylor, Debbie Hensgen, Richard Freund, and Lantz Moore, "*SmartNet: A Scheduling Framework for Heterogeneous Computing,*" 2nd IEEE International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN '96), sponsored by the IEEE Computer Society, Beijing, China, June 1996, pp.514-521.

[KIDD99] Kidd, Taylor W. "Personal Notes," Unpublished, 1999.

[KILG98] Kilgore, Richard and Kevin Healy. "*Introduction to Silk and JAVA-based Simulation,*" [http://www.threadtec.com/papers/paper2/paper2.html]. 1998.

[LAW91] Law, Averill M. and W.David Kelton. Simulation Modeling and Analysis, 2nd Edition. New York: McGraw-Hill, 1991.

[MAHE99] Maheswaran, Muthucumaru, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund, "*Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems,*" Proceedings of the 8th IEEE Workshop on Heterogeneous Computing Systems (HCW'99), San Juan, Puerto Rico, pp. 30-44, April 1999.

[PORT99] Porter, N. W. "*The Need for Adaptive and Adaptation-Aware C4I Models in a Distributed Heterogeneous Computing Environment,*" Master's Thesis. Naval Postgraduate School, Monterey, California, June 1999.

[RUBI81] Rubinstein, Reuven Y. Simulation and the Monte Carlo Method. New York: Wiley, 1981.

[SCHN98] Schnaidt, Matthew C. "*Design, Implementation, and Testing of MSHN's Application Resource Monitoring Library,*" Master's Thesis. Naval Postgraduate School, Monterey, California, December 1998.

[SING94] Singhal, Mukesh and Niranjan G. Shivaratri. Advanced Concepts in Operating Systems. New York: McGraw-Hill, 1994.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Road
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center 2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

5. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   3300 Russell Road
   Quantico, VA 22134-5130

6. Marine Corps Representative 1
   Naval Postgraduate School
   Code 037, Bldg. 234, HA-220
   699 Dyer Road
   Monterey, CA 93940

7. Marine Corps Tactical Systems Support Activity.....1
   Technical Advisory Branch
   Attn: Maj J. C. Cummiskey
   Box 555171
   Camp Pendleton, CA 92055-5080

8. Chairman,Code CS.....1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93940-5000

9. Dr. Taylor Kidd.....1
Computer Science Department, Code CS/Kt
Naval Postgraduate School
Monterey, California 93943-5100

10. Dr. Debra Hensgen.....1
Computer Science Department, Code CS/Hd
Naval Postgraduate School
Monterey, California 93943-5100

11. Major James M. Breitinger 2
1481 Riverwood Lane
Phoenixville, PA 19460